Universidade de Brasília Faculdade UnB Gama

Aluna: Tatielen Rodrigues Dutra Pereira
Matricula: 12/0136074
Data: 21/11/2017

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 ;
    JGE divisao_subtract ;
    CLR.W R15 ;     RET
divisao_subtract:
    SUB.W R14,R15 ;
    PUSH.W R13 ;
    CALL #divisao_unsigned ;
    POP.W R14 ;
    CLR.W R14;
    CLR.W R13;
    RET
Calcula a raiz quadrada de R15, um inteiro.
raiz_quadrada:
    MOV.W #0,R14 ;
    TST R14;
    JNE else_iteracao;
    PUSH R15 ;
    MOV.W #2,R14 ;
    CALL #divisao_unsigned ;
    MOV.W R15,R14 ;
    POP R15 ;
    CALL #realizar_iteracao ;
    JMP condicao_while ;
realizar_iteracao:
    PUSH R15;
    PUSH R14;
    CALL #divisao_unsigned ;
    MOV.W R15,R13 ;
    POP R14 ;
    ADD.W R14,R13 ;
    MOV.W R13,R15 ;
    MOV #2,R14 ;
    CALL #divisao_unsigned
```

```
        MOV R15,R14 ;
        POP R15 ;
        RET

else_iteracao:
        CALL #realizar_iteracao

condicao_while:
        PUSH R14 ;
        PUSH R14 ;
        CALL #realizar iteracao
        MOV.W R14,R13 ;
        POP R14 ;
        SUB.W R13,R14 ;
        MOV.W R14,R13 ;
        POP R14 ;
        CMP #1, R13 ;
        JGE 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);

{
        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"

    NAME    main
    PUBLIC  main

    ORG     0FFFEh
    DC16    init
    RSEG    CSTACK                  RSEG    CODE
init:   MOV     #SFE(CSTACK), SP

main:   NOP                         MOV.W   #WDTPW+WDTHOLD,&WDTCTL
    MOV.W   #0xFFFB,R15                 MOV.W   #3,R14
    CALL    #Potencia
    JMP $

MULT_signed:

    PUSH    R15
    RLA     R15                                 JNC     Third_else_if_MULT_signed
POP     R15
    PUSH    R14
```

```
    RLA    R14                          JNC    Second_else_if_MULT_signed
POP    R14
    INV    R15                          INC    R15
INV    R14
    INC    R14
    CALL   #MULT_unsigned
    CLR    R14
    RET

Second_else_if_MULT_signed:                     POP    R14
    INV    R15                          INC    R15
    CALL   #MULT_unsigned
    INV    R15                          INC    R15
CLR    R14
    RET

Third_else_if_MULT_signed:                      POP    R15
    PUSH   R14
    RLA    R14                          JNC    else__MULT_signed
POP    R14
    INV    R14                          INC    R14
CALL   #MULT_unsigned
    INV    R15                          INC    R15
    CLR    R14
    RET

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

MULT_unsigned:

    TST    R14
    JNZ    MULT_unsigned_else
    CLR.W  R15
    RET

MULT_unsigned_else:

    PUSH   R15
    DEC.W  R14
    CALL   #MULT_unsigned
    POP.W  R14
    ADD.W  R14,R15
```

```
    RET

Potencia:

    TST    R14
    JNZ    Potencia_else
    MOV.W  #1,R15
    RET

Potencia_else:

    PUSH   R15
    DEC.W  R14
    CALL   #Potencia
    POP.W  R14
    CALL   #MULT_signed
    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                          JNC    Third_else_if_DIV_signed
    POP    R15
    PUSH   R14
    RLA    R14                          JNC    Second_else_if_DIV_signed
POP    R14
    INV    R15                          INC    R15
INV    R14
    INC    R14
    CALL   #divisao_unsigned
    CLR    R14
    RET
Second_else_if_DIV_signed:                      POP    R14
    INV    R15                          INC    R15
    CALL   #divisao_unsigned
    INV    R15                          INC    R15
CLR    R14
    RET

Third_else_if_DIV_signed:                       POP    R15
```

```
    PUSH   R14
    RLA    R14                          JNC    else_DIV_signed
POP    R14
    INV    R14                          INC    R14
    CALL   #divisao_unsigned
    INV    R15                          INC    R15
CLR    R14
    RET

else_DIV_signed:
    POP    R14
    CALL   #divisao_unsigned                    CLR    R14
    RET

divisao_unsigned:
    MOV #1,R13
    CMP R14,R15
    JGE divisao_subtract
    CLR.W R15
    RET

divisao_subtract:
    SUB.W R14,R15
    PUSH.W R13
    CALL #divisao_unsigned
    POP.W R14
        ADD.W R14,R15
    CLR.W R14
    CLR.W R13
    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
    SUB.W R14,R15
    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 Primalidade(unsigned int x);

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

        return dividend;
}

int Primalidade (int a)
{
        int i=3;
        if (a==1)
        {
                return 0;
        }
        else if (a==2)
        {
                return 1;
        }

        else if (Remainder(a,2)==0)
        {
                return 0;
        }

        else
```

```
        {
                while(Remainder(a,i)!=0 && i<a)
                {
                        i+=2;
                }
                                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
    JEQ    Nao_primo
    CMP    #1,R15
    JEQ    Nao_primo
    CMP    #2,R15
    JEQ    E_primo
    MOV.W  #3,R14
While_verificar:
    CMP    R14,R15
    JEQ    E_primo
    PUSH   R14
    PUSH   R15
    CALL   #remainder
    TST    R15
    JEQ    Nao_primo_2
    POP    R15
    POP    R14
    ADD.W  #2,R14
    JMP    While_verificar
Nao_primo:
    MOV.W  #0,R15
    CLR.W  R14
    RET

Nao_primo_2:
```

```
    POP    R15
    POP    R15
    MOV.W  #0,R15
    CLR.W  R14
    RET
E_primo
    MOV.W  #1,R15
    CLR.W  R14
    RET

remainder:
    CMP    R14,R15
    JL     remainder_finish
    SUB.W  R14,R15
    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!! = 135*...n, e se n for par, n!! = 246...n. Por exemplo, 9!! = 13579 = 945 e 10!! = 2468*10 = 3840. Além disso, 0!! = 1!! = 1. O protótipo da função é:

```
unsigned long long DuploFatorial(unsigned long long n);
```

```
#include<stdio.h>
{
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;
	}

	else if(Remainder(n,2)==0)
	{

		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;
}
```

(b) Escreva a sub-rotina equivalente na linguagem Assembly do MSP430, mas considere que os valores de entrada e de saída são inteiros de 16 bits. A variável de entrada é fornecida pelo registrador R15, e o valor de saída também.

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.

```
int Vetor_Ordenado_Decrescente(int *p,int n)
{
	int i,anterior=0,proximo=0;
	for (i=0;i<n;i++)
	{
		if(i==0)
		{
			anterior = p[i];
}
		else
		{
			proximo = p[i];
```

```
                        if(anterior>proximo)
                        {
                                anterior = proximo;
                        }
                        else
                        {
                                return 0;
                        }

                }
        }
                return 1;
}
```

Agora para MSP 430:

```
#include "msp430.h"
#include <msp430g2553.h>
      NAME    main

      PUBLIC  main

      ORG     0FFFEh
      DC16    init
      RSEG    CSTACK
      RSEG    CODE

init:   MOV     #SFE(CSTACK), SP

main:   NOP
      MOV.W   #WDTPW+WDTHOLD,&WDTCTL
      MOV.W   #0x0A30, 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

CALL #Vetor_Ordenado_Decres
      JMP $
 Vetor_Ordenado_Decres:
      MOV.W   #0,R13
 For_vetor_ord_decres:
      TST     R13
      JEQ     Fim_Iteracao_for_ord_decres
```

```
    CMP     R14,R13
    JGE     end_for_vetor_ord_decres
    MOV.W   R13,R12
    SUB.W   #2,R12
    PUSH    R13
    ADD.W   R15,R13
    ADD.W   R15,R12
    CMP     0(R12),0(R13)
    JGE     Nao_decrescente
    POP     R13

Fim_Iteracao_for_ord_decres:
    ADD     #2,R13
    JMP     For_vetor_ord_decres

Nao_decrescente:
    POP     R15
    CLR     R15
    CLR     R14
    CLR     R13
    CLR     R12
    RET

end_for_vetor_ord_decres:
    MOV.W   #1,R15
    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)
{

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

Em Assembly:

```
#include "msp430.h"

    NAME    main

    PUBLIC  main

    ORG     0FFFEh
```

```
      DC16    init
label

      RSEG    CSTACK
            RSEG    CODE

init:   MOV     #SFE(CSTACK), SP

main:   NOP
      MOV.W   #WDTPW+WDTHOLD,&WDTCTL
      MOV.W   #0x0a10,R15
      MOV.W   #0x0a20,R14
      MOV.W   #0xFFFF,0(R15)
      MOV.W   #0xFFFF,2(R15)
      MOV.W   #0xFFFF,4(R15)
      MOV.W   #1,0(R14)
      MOV.W   #1,2(R14)
      MOV.W   #1,4(R14)
      MOV.W   #6,R13
      CALL    #Produto_Escalar

      JMP $


Produto_Escalar:
;-------------------------------------------------------------------------
; R15 = a
; R14 = b
; R13 = n
; R12 = i
; R11 = soma
;-------------------------------------------------------------------------
      CLR     R12
      CLR     R11
For_Produto_Escalar:
      CMP     R13,R12
      JGE     end_for_produto_escalar
      PUSH    R15
      PUSH    R14
      ADD.W   R12,R15
      ADD.W   R12,R14


      MOV.W   0(R15),R15
      MOV.W   0(R14),R14
      CALL    #MULT_signed
```

```
    ADD.W   R15,R11
        POP     R15
    POP     R14
    INCD    R12
    JMP     For_Produto_Escalar


end_for_produto_escalar:

    MOV.W   R11,R15
    CLR     R14
    CLR     R13
    CLR     R12
    CLR     R11
    RET

MULT_signed:

    PUSH    R15
    RLA     R15                         JNC     Third_else_if_MULT_signed
    POP     R15
    PUSH    R14
    RLA     R14                         JNC     Second_else_if_MULT_signed
    POP     R14
    INV     R15                         INC     R15
INV     R14
    INC     R14
    CALL    #MULT_unsigned
    CLR     R14
    RET
Second_else_if_MULT_signed:                     POP     R14
    INV     R15                         INC     R15
    CALL    #MULT_unsigned
    INV     R15                         INC     R15
CLR     R14
        RET

Third_else_if_MULT_signed:                      POP     R15
    PUSH    R14
    RLA     R14                         JNC     else__MULT_signed
POP     R14
    INV     R14                         INC     R14
CALL    #MULT_unsigned
    INV     R15                         INC     R15
CLR     R14
    RET
```

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

MULT_unsigned:

    TST    R14
    JNZ    MULT_unsigned_else
    CLR.W  R15
    RET

MULT_unsigned_else:

    PUSH   R15
    DEC.W  R14
    CALL   #MULT_unsigned
    POP.W  R14
    ADD.W  R14,R15
    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 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"                file

      NAME    main

      PUBLIC  main

            ORG     0FFFEh
      DC16    init
            RSEG    CSTACK
      RSEG    CODE

init:   MOV     #SFE(CSTACK), SP

main:   NOP
      MOV.W   #WDTPW+WDTHOLD,&WDTCTL
      MOV.W   #0x0a00,R15
            MOV.W   #1,0(R15)
      MOV.W   #2,2(R15)
      MOV.W   #3,4(R15)
            MOV.W   #2,6(R15)
      MOV.W   #1,8(R15)
      MOV.W   #10,R14
      CALL    #Palindromo
      JMP $

Palindromo:
      CLR     R13
For_Palindromo:
      CMP     R14,R13
      JGE     End_For_Palindromo
      PUSH    R13
      PUSH    R14
      DECD    R14
      SUB.W   R13,R14
      ADD.W   R15,R14
      ADD.W   R15,R13
      CMP     0(R13),0(R14)
            JNE     Nao_palindromo
      POP     R14
      POP     R13
      INCD    R13
      JMP     For_Palindromo
```

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

Nao_palindromo:
      POP     R15
      POP     R15
      CLR     R15
      CLR     R14
      CLR     R13
      RET

      END
```