# MC404: Organização de Computadores e Linguagem de Montagem

2ª Lista de Exercícios (24/08/2018)

**Questão 1.** Traduza os trechos de programa em C abaixo para linguagem de montagem do ARM.

**A)**

```
char  c, *pc;
int i, *pi;
...

c = 'a';
pc = &c;
*pc = 'c';

i = -1;
pi = &i;
*pi = 4096;
```

**Solução**

```
@ área de dados
@ reserva e inicialização de espaço para as variáveis

i:      .skip 4         @ int i
pc:     .skip 4         @ char *pc
pi:     .skip 4         @ int *pi
c:      .skip 1         @ char c

        .align 2
@ área de código

        mov     r0,#0x61        @ c = 'a'
        ldr     r1,=c
        strb    r0,[r1]

        ldr     r2,=pc          @ pc = &c
        str     r1,[r2]

        mov     r0,#0x63        @ *pc = 'c'
                                @ reaproveitando r1
        strb    r0,[r1]         @ r1 é apontador de byte

        ldr     r1,=i           @ i = -1
        mov     r0,#-1
        str     r0,[r1]

        ldr     r2,=pi          @ pi = &i
        str     r1,[r2]         @ r1 contém o endereço de i

        mov     r0,#4096        @ *pi = 4096
                                @ reaproveitando r1
        str     r0,[r1]         @ r1 é apontador de palavra
```

**B)**

```
int x=1;
char vet[100],*p;
...
p = &vet[0];
*p = (char) x; // considere dois casos: big-endian e little endian
```

**Solução**

```
@ área de dados
@ reserva e inicialização de espaço para as variáveis

x:      .word 1         @ int x = 1
vet:    .skip 100       @ char vet[100]
p:      .skip 4         @ char *p

        .align 2
@ área de código
        ldr     r1,=p
        ldr     r2,=vet         @ p = &vet[0]
        str     r2,[r1]

        ldr     r3,=x           @ *p = (char) x
        ldrb    r0,[r3]         @ valor de x
        str     r0,[r2]         @ reaproveitando r2
```

**Questão 2.** Escreva um trecho de programa, em linguagem de montagem do ARM, que considerando que `r1` contém o endereço inicial de uma cadeia de caracteres, composta por letras na representação ASCII e espaços em branco (caractere `0x20`), terminada pelo caractere `0x00`, coloque em `r0` o número de espaços em branco que a cadeia contém.

**Questão 3.** Escreva um trecho de programa, em linguagem de montagem do ARM, que considerando que `r1` contém o endereço inicial de uma cadeia de caracteres, composta por letras na representação ASCII e espaços em branco(caractere `0x20`), terminada pelo caractere `0x00`, substitua todas as ocorrências de dois ou mais espaços em branco por apenas um espaço em branco. Por exemplo, se a cadeia é

```
␣␣␣␣␣Uma␣␣␣␣␣cadeia␣␣␣␣␣␣␣bem␣simples
```

seu programa deve transformá-la da cadeia

```
␣Uma␣cadeia␣bem␣simples
```

A cadeia deve ser transformada "in place", ou seja, não devem ser usadas outras regiões de memória além da que a cadeia já ocupa.

**Solução**

```
@ área de código
        .global _start
_start:
        ldr     r2,=cadeia      @ vamos usar dois registradores
        mov     r3,r2           @ para percorrer a cadeia
```

```
                                @ vamos construir a nova cadeia "in place"
                                @ r2 aponta para o final corrente da "nova" cadeia
                                @ r3 aponta para o final corrente da cadeia original
        ldrb    r0,[r3]         @ carrega um caractere da cadeia original
        mov     r5,#0           @ r5 vai marcar se já encontramos um espaço em branco
                                @ r5 = 0 não encontrou ainda
loop:
        cmp     r0,#0           @ verifica se terminou a cadeia
        beq     finaliza
        cmp     r0,#0x20        @ verifica se caractere da cadeia original é espaço
        beq     espaco          @ desvia se é espaço
        strb    r0,[r2]         @ não é espaço, armazena caractere na nova cadeia
        mov     r5,#0           @ marca que não temos espaço ainda na nova cadeia
        add     r2,#1           @ avança apontador da nova cadeia
        b       avanca
espaco:
        cmp     r5,#0           @ verifica se já temos um espaço na nova cadeia
        bne     avanca          @ desvia se já temos, não pode colocar outro
        strb    r0,[r2]         @ não temos, coloca caractere espaço na nova cadeia
        add     r2,#1           @ avança apontador da nova cadeia
        mov     r5,#1           @ marca que já temos espaço na nova cadeia
avanca:
        add     r3,#1           @ avança apontador da cadeia original
        ldrb    r0,[r3]         @ carrega próximo caractere
        b       loop            @ e continua no loop
finaliza:
        strb    r0,[r2]
                                @ e terminou
@ apenas para testar no armsim
        mov     r0,#0         @ status in r0 (0 means no errors)
        mov     r7,#1         @ exit is syscall #1
        swi     #0x55         @ invoke syscall

@ área de dados
cadeia:
        .asciz    "   um   teste     simples    "
```

**Questão 3.** Escreva um trecho de programa em linguagem de montagem do ARM que implemente a soma de dois vetores $vetA$ e $vetB$ de números inteiros de 32 bits sem sinal, armazenando o resultado no vetor $vetC$. Assuma que os inícios dos vetores $vetA$, $vetB$ e $vetC$ estão respectivamente nos endereços indicados pelos rótulos `vetA`, `vetB` e `vetC`, e que o número de elementos do vetor (mesmo para todos os vetores) é dado em uma palavra de memória armazenada no endereço de rótulo `num_elem` (ou seja, o número de elementos é o conteúdo da palavra cujo rótulo é `num_elem`).

**Solução**

```
@ área de código
        .global _start
_start:
        ldr     r1,=vetA        @ apontador para vetA
        ldr     r2,=vetB        @ apontador para vetB
        ldr     r3,=vetC        @ apontador para vetC
        ldr     r4,=num_elem
```

```
        ldr     r4,[r4]         @ número de elementos em r4
loop:
        ldr     r5,[r1]         @ elemento de vetA
        ldr     r6,[r2]         @ elemento de vetB
        add     r5,r6           @ adiciona os dois elementos
        str     r5,[r3]         @ armazena resultado como elemento de vetC
        add     r1,#4           @ avança apontador
        add     r2,#4           @ avança apontador
        add     r3,#4           @ avança apontador
        subs    r4,#1           @ tratamos mais um elemento
        bne     loop
                                @ e termina
@ apenas para testar no armsim
        mov     r0,#0           @ status in r0 (0 means no errors)
        mov     r7,#1           @ exit is syscall #1
        swi     #0x55           @ invoke syscall

@ área de dados
vetA:
        .word 1,2,3,4
vetB:
        .word 5,6,7,8
vetC:
        .word 9,10,11,12
num_elem:
        .word 4
```

**Questão 4.** Altere o trecho de programa da questão anterior para zerar todos os elementos do vetor resultado ($vetC$) caso ocorra estouro de campo na soma de algum dos elementos.