PUC-Rio – Software Básico – INF1612
Prova 1 – 16/04/05

1. (3,0 pontos) Considere o programa C a seguir (as reticências na inicialização de S são propositais):

```
#include <stdio.h>
void dump (void *p, int n) {
  unsigned char *p1 = (unsigned char *) p;
    while (n--) {
      printf("%p - %02x\n", p1, *p1);
      p1++;
   }
}
struct s {
  char c;
  int i;
  short s;
} S[2] = {{...},{...}};
int main (void) {
  dump (&S, 30);
  return 0;
}
```

Imagine que, ao ser executado em uma máquina *little-endian*, esse programa imprimiu a saída a seguir:

```
0x80494e4 - 32
0x80494e5 - 00
0x80494e6 - 01
0x80494e7 - 00
0x80494e8 - fe
0x80494e9 - 03
0x80494ea - 00
0x80494eb - 00
0x80494ec - fb
0x80494ed - ff
0x80494ee - 00
0x80494ef - ff
0x80494f0 - 63
0x80494f1 - 00
0x80494f2 - f0
0x80494f3 - 00
0x80494f4 - ff
0x80494f5 - f7
0x80494f6 - ff
0x80494f7 - ff
0x80494f8 - 43
0x80494f9 - 00
0x80494fa - 00
0x80494fb - 00
0x80494fc - 01
0x80494fd - 00
0x80494fe - 20
0x80494ff - 00
0x8049500 - 01
0x8049501 - 00
```

Analisando a saída e a declaração de `struct s`, diga quais foram os valores atribuídos a `S` na sua inicialização. (ATENÇÃO: o programa pede que dump imprima 30 bytes! Esse número é

suficiente para mostrar o array `S`, mas podem ser impressos bytes a mais.) Expresse os valores em decimal, no caso de `int` e `short`, e em caracter ascii, no caso de `char`. **Explique** como você chegou aos valores (mostre suas contas e a posição relativa dos dados!!). Suponha que a máquina de execução é *little-endian*. O valor ascii de 'a' é 0x61 e o de '0' 0x30.

2. (3,0 pontos) Escreva um trecho de código assembly (IA32, sintaxe gcc/Linux) que imprima o produto interno de dois vetores de 10 posições declarados como globais, como abaixo:

```
.data
a:   .int  5, -3, 4, 12, -99, 50, 21, 8, -2, 100
b:   .int  -3, -2, 1, -2, 0, 0, 2, 1, 2, -1
```

   (O produto interno de dois vetores `a[10]` e `b[10]` é dado por $\sum_{i=0}^{9} a[i] \cdot b[i]$.)

   Para imprimir o valor calculado, chame a função *printnum*, usada nos laboratórios.

3. (4,0 pontos) Suponha que foi declarado um array global de bytes de nome `a`. Cada byte do array contém, nos 7 bits menos significativos, um código de caracter `ascii`, e, no bit mais significativo, um bit de *paridade*, cujo valor deve ser 1 se o número de bits iguais a 1 nos demais 7 bits for par e 0 se o número de bits iguais a 1 nos demais 7 bits for ímpar. Escreva uma função C ou um trecho de código assembly (IA32, sintaxe gcc/Linux) que verifique a *paridade* dos bytes nesse array e calcule o número de bytes encontrados com bit de paridade errado. Caso você escreva a função em C, retorne esse número calculado como resultado da função. Caso você escreva um trecho de código assembly, deixe-o em `%eax`.

   Exemplos:

   10010001 bit de paridade correto (2 bits com valor 1 nos 7 bits menos significativos)

   00010001 bit de paridade errado (2 bits com valor 1 nos 7 bits menos significativos)

   00011001 bit de paridade correto (3 bits com valor 1 nos 7 bits menos significativos)