PUC-Rio – Software Básico – INF1018
Prova 1 – Turma 3wb – 2/10/2015

1. (2,5 pontos) Considere o programa C a seguir:

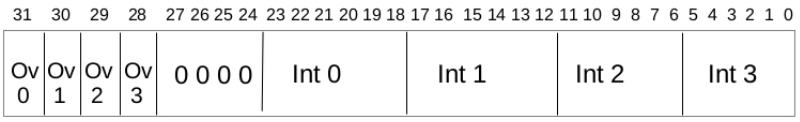
```
#include <stdio.h>
void dump (void *p, int n) {
  unsigned char *p1 = (unsigned char *) p;
  while (n--) {
    printf("%p - %02x\n", p1, *p1);
    p1++;
  }
}
struct X {
  struct X *prox;
  unsigned char flags;
  int val1;
  short val2;
};
struct X nos[4];

int main (void) {
  nos[0].prox = &nos[1];
  nos[0].flags = (5 << 3 ) & 160;
  nos[0].val1 = -1027;
  nos[0].val2 = 133;
  dump(nos, sizeof(struct X));
  return 0;
}
```

   Supondo que `nos` seja alocado na posição de memória 0x0804a040, diga o que esse programa irá imprimir quando executado. Coloque **PP** nas posições correspondentes a *padding*.

   Considere que a máquina de execução é *little-endian* e que as convenções de alinhamento são as do Linux no IA-32 (vistas em sala).

   (ATENÇÃO: mostre como você chegou aos valores exibidos. Valores sem contas **NÃO** valem ponto!).

2. (1,5 pontos) Considere o vetor de pequenos inteiros do nosso primeiro trabalho. Representamos esse vetor utilizando um inteiro sem signal com o seguinte formato:



   - os bits 31, 30, 29 e 28 indicam a ocorrência de *overflow* na geração dos valores armazenados nos campos Int0 a Int3.
   - os bits 27 a 24 contém o valor 0 (zero)
   - os bits 23-18 (Int0), 17-12 (Int1), 11-6 (Int2) e 5-0 (Int3) armazenam os pequenos inteiros (valores inteiros de 6 bits, em complemento a 2).

Implemente em C uma função `vs_overflow`, que indica se o valor armazenado no pequeno inteiro indicado sofreu truncamento com perda de valor (isto é, se o bit de *overflow* correspondente está "setado"). A função pedida tem o protótipo:

```
typedef unsigned VetSmallInt;
int vs_overflow(VetSmallInt v, int n);
```

Ela recebe como parâmetros um vetor de pequenos inteiros (`v`) e um inteiro que indica qual o pequeno inteiro em questão (`n`, que pode assumir valores de 0 a 3) e retorna 1 se o bit de *o*verflow correspondente a este pequeno inteiro está "setado", ou 0 caso contrário.

3. Traduza as funções `foo` e `boo` abaixo para assembly IA-32 (o assembly visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros, salvamento de registradores e resultados em C/linux. Traduza o mais diretamente possível o código de C para assembly.

   (Não se preocupe em entender o que as funções fazem, apenas traduza-as literalmente.)

   Comente seu código!

   (a) (2,5 pontos)

```
void foo (int a[], int b[], int n) {
  int i, j;
  for (i = 0, j = n-1; i < n; i++, j--)
    b[i] = a[j];
}
```

   (b) (2,5 pontos)

```
struct X {
  int val1;
  int val2;
};

int boo (struct X *px, int n) {
  if (n == 0)
    return 0;
  if (px->val1 == 0)
    return px->val2;
  else
    return boo(px+1, n-1);
}
```

Boa Prova!