PUC-Rio — Software Básico — INF1612
P1 — 03/10/2006

1. Considere uma representação de grandes números inteiros via um array, que deve ser interpretado como um único inteiro de 80 bits, em complemento a dois, seguindo a ordem little-endian:

```
#define NUM_BYTES 10
typedef unsigned char BigInt[NUM_BYTES];
```

   (a) (2,0) Escreva uma função para somar dois números nesse formato, seguindo o protótipo abaixo:

```
/* res = a + b */
void bi_sum (BigInt res, BigInt a, BigInt b);
```

   (b) (2,0) Escreva uma função para calcular o complemento a dois de um dado número, seguindo o protótipo abaixo. (Se necessário, pode usar a função `bi_sum`, mesmo sem tê-la definido.)

```
/* res = -a */
void bi_cp2 (BigInt res, BigInt a);
```

2. (3,0) Considere o código abaixo:

```
struct X {
  int v;
  struct X *next;
};
int f (int x);
int add (struct X *x) {
  int a = 0;
  for (; x != NULL; x = x->next)
    a += f(x->v);
  return a;
}
```

   Traduza a função `add` para assembly IA-32 (o assembly visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros e retorno em C/Linux. Assuma que a função `f` já está definida. Comente seu código.

3. (3,0) 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 um {
  int f;
  char c;
  short d;
  char i;
};
struct um a =  {(0x0aaa & -23), ~(12 << 4), (0x0f | 1000), (~(-4) + 1)};
int main (void) {
  dump (&a, sizeof(a));
  return 0;
}
```

   Considerando que `a` seja alocado na posição de memória `0xbffff00c`, diga o que esse programa irá imprimir quando executado, explicando como você chegou aos valores exibidos (*mostre suas contas!*). Suponha que a máquina de execução é Pentium-Linux (inteiros de 32 bits, little-endian), ou seja, a representação de dados é a vista no curso.