PUC-Rio – Software Básico – INF1018
Prova 2 – 27/11/2014

1. (2,0 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 {
  short s;
  float f;
  double d;
} x = {-132, -7.625, 2048+512};

int main(void) {
  dump((void *)&x, sizeof(x));
  return 0;
}
```

   Supondo que x seja armazenado no endereço de memória 0x804966c, diga o que o programa irá imprimir quando executado, deixando claro como você chegou a esses valores. Considere que a máquina de execução é *little-endian*, e que as convenções de alinhamento são as do Linux no IA-32. Se houver posições de *padding*, indique seu conteúdo com **PP**. (ATENÇÃO: valores sem contas e explicações **NÃO** valem ponto!)

2. (1,5 pontos) Considere o seguinte arquivo *assembly* `arq.s`:

```
.data
.globl valor
valor:  .int  0

.text
sc: .string "%d\n"

.globl funcaoA
funcaoA:
  push %ebp
  movl %esp, %ebp
  push $valor
  push 8(%ebp)
  call funcaoB
  addl $8, %esp
  push %eax
  push $sc
  call printf
  addl $8, %esp
  movl %ebp, %esp
  popl %ebp
  ret
```

   Suponha que o arquivo `arq.s` seja processado pelo montador, gerando um arquivo objeto `arq.o` e que esse arquivo objeto seja dado como entrada para o programa `nm`. Liste quais símbolos

apareceriam como **D** (símbolo na área de dados, exportado), **T** (símbolo na área de código, exportado) e **U** (símbolo indefinido).

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

   (a) (2,5 pontos)

```
#define TAM 5
void f (int *pi, int i);
int g (int *vi, int i);
int foo (int val) {
  int local[TAM]; int i;
  for (i=0;i<TAM;i++)
    f(&local[i], val);
  return g(local, TAM);
}
```

   (b) (2,0 pontos)

```
float bar (double v, double* a, int n) {
  float f = 1.0;
  while (n--) {
    f *= *a + v + n;
    a++;
  }
  return f;
}
```

4. (2,0 pontos) Suponha que compilamos e ligamos um determinado programa `chama.c` onde a `main` contém uma chamada a uma função `moo`, gerando um executável `chama`. Ao executar `objdump -d chama`, um dos trechos resultantes é o seguinte (com uma pequena modificação explicada a seguir):

```
08048374 <moo>:
 8048374:       55                      push   %ebp
 8048375:       89 e5                   mov    %esp,%ebp
 8048377:       b8 01 00 00 00          mov    $0x1,%eax
 804837c:       5d                      pop    %ebp
 804837d:       c3                      ret

0804837e <main>:
 804837e:       8d 4c 24 04             lea    0x4(%esp),%ecx
 8048382:       83 e4 f0                and    $0xfffffff0,%esp
 8048385:       ff 71 fc                pushl  0xfffffffc(%ecx)
 8048388:       55                      push   %ebp
 8048389:       89 e5                   mov    %esp,%ebp
 804838b:       51                      push   %ecx
 804838c:       e8 HH HH HH HH          call   <moo>
 8048391:       59                      pop    %ecx
 8048392:       5d                      pop    %ebp
 8048393:       8d 61 fc                lea    0xfffffffc(%ecx),%esp
 8048396:       c3                      ret
```

   Neste trecho, substituímos uma sequência de quatro bytes originalmente exibidos em hexadecimal pela sequência "HH HH HH HH". Calcule os valores hexadecimais da sequência original.

   obs: Lembre-se que, na saída do `objdump`, a coluna à esquerda representa o endereço de memória onde está cada instrução, a coluna do meio exibe os códigos de máquina contidos no executável, e a coluna da direita contém o provável código assembly que gerou esse código de máquina.