PUC-Rio – Software Básico – INF1018
Prova 2 – Turma 3wb – 27/06/2019

1. (1,5 pontos) Considere o arquivo `mod.c` abaixo e, a seu lado, o *desassembly* do objeto `mod.o`, produzido por uma compilação desse arquivo:

```
#include <math.h>
#define TAM 4
extern int x;
int f3(int i);

int y = 1024;

double f1(double a, double b) {
  int n = TAM;
  while (n--) {
    a = a + b;
  }
  return sin(a);
}

int f2() {
  return f3(x);
}
```

```
0000000000000000 <f1>:
   0: b8 04 00 00 00        mov    $0x4,%eax
0000000000000005 <l1>:
   5: f2 0f 58 c1           addsd  %xmm1,%xmm0
   9: ff c8                 dec    %eax
   b: 75 f8                 jne    5 <l1>
   d: e8 00 00 00 00        callq  12 <l1+0xd>
  12: c3                    retq
0000000000000013 <f2>:
  13: 8b 3c 25 00 00 00 00  mov    0x0,%edi
  1a: e8 00 00 00 00        callq  1f <f2+0xc>
  1f: c3                    retq
```

   (a) Liste quais símbolos de `mod.c` são `definidos E exportados` por este módulo (ou seja, os símbolos visíveis por outros módulos).

   (b) Liste quais símbolos são `referências externas`.

   (c) Na listagem do *desassembly*, os "offsets" das instruções (seguidos por um ":") são indicados em todas as linhas. Indique os offsets das linhas onde há instruções que deverão ser modificadas pelo linkeditor durante a etapa de *relocação de endereços*.

2. (2,5 pontos) Considere o programa C a seguir, executado em uma máquina IA-64, com ordenação *little-endian*, e respeitando as convenções de alinhamento para Linux nesta plataforma:

```
#include <stdio.h>
void dump (void *p, int n) {
  unsigned char *p1 = p;
  while (n--) {
    printf("%p - %02x\n", p1, *p1);
    p1++;
  }
}
struct X {
  short s[2];
  double d;
  float f;
} x = {{-4095, 18}, 13.75, 13.75*2};

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

Sabendo que o array `x` está armazenado no endereço de memória `0x601060`, e que a máquina de execução é *little-endian*, com as convenções de alinhamento do Linux no IA-64, mostre o que o programa irá imprimir quando executado. Coloque **PP** nas posições correspondentes a *padding*.

(Deixe claro como você chegou aos valores exibidos! Valores sem contas não serão considerados.)

3. Traduza as funções `foo` e `boo` a seguir para assembly IA-64 (visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros e retorno de resultado em C/Linux.

   (a) (2,5 pontos)

```
struct X {
  char   v1;
  float  v2;
  double v3;
  struct X *next;
};
double foo1(double x);

void foo(struct X *px, float v) {
  while (px) {
    if (px->v1)
      px->v3 = foo1(px->v2);
    else
      px->v3 = px->v2 + v;
    px = px->next;
  }
}
```

   (b) (2,5 pontos)

```
#define VTAM 20
void boo1(int x, int *p);
int boo2(int x, int *y, int z);

int boo(int x) {
  int v[VTAM];
  int i;

  for (i = 0; i < VTAM; i++)
    boo1(i, &v[i]);

  return boo2 (x, v, VTAM);
}
```

4. (1,0 ponto) Escreva o código **assembly** equivalente ao código de máquina que sua função *gera* do segundo trabalho geraria como tradução da função *Simples* a seguir:

```
v1 < p1
v2 < $2
v1 = v1 * v2
ret v1
```