PUC-Rio – Software Básico – INF1018
Prova 2 – 3/12/2009

1. (3,5 pontos) Traduza a função `bar` abaixo para assembly IA-32 (o assembly visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros e retorno em C/Linux. Não se preocupe com o que a função faz, apenas a traduza. Comente seu código.

```
double a = 2.0;
float doo (float x);

int bar (double *nums, int n) {
  int i;
  double total = 0.0;
  for (i=0;i<n;i++) {
   total = total + doo(nums[i]);
  }
  return (int) (total+ a);
}
```

2. (2,5 pontos) Dado o seguinte código na linguagem `SB` do segundo trabalho:

   (não se preocupe em extrair significado lógico da função!!!)

   (a) (2 pontos) Escreva o código assembly equivalente, como seria criado pelo seu procedimento *Compila*. Use *assembly* e use labels simbólicos para as chamadas. (*Traduza como seu programa o faria, e não como você escreveria uma programa assembly normal!*

   (b) (0,5 ponto) Supondo que o retorno de *Compila* fosse atribuído a uma variável f, diga qual seria o resultado da chamada f(98).

```
function
v[0]=c[100]-p[0]
ret?v[0]
v[0]=p[0]+c[1]
call0v[0]
v[0]=p[0]+v[1]
ret?c[0]
end
```

3. (2,0 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 {
  float f;
  char c;
  double d;
} S = {...};
int main (void) {
  dump (&S, 20);
  return 0;
}
```

Ao ser executado em uma máquina Linux IA-32, esse programa imprimiu a saída a seguir:

```
0x80495dc - 00
0x80495dd - 00
0x80495de - 98
0x80495df - c0
0x80495e0 - 61
0x80495e1 - 00
0x80495e2 - 00
0x80495e3 - 00
0x80495e4 - 00
0x80495e5 - 00
0x80495e6 - 00
0x80495e7 - 00
0x80495e8 - 60
0x80495e9 - 00
0x80495ea - f0
0x80495eb - 40
0x80495ec - 00
0x80495ed - 00
0x80495ee - 00
0x80495ef - 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 20 bytes! Esse número é suficiente para mostrar `S`, mas podem ser impressos bytes a mais.) Expresse os valores sempre na base 10. **Explique** como você chegou aos valores (mostre suas contas e a posição relativa dos dados!!). Suponha que a máquina de execução é *little-endian*.

4. (2,0) Considere um arquivo contendo o código abaixo:

```
#include <math.h>
extern double a[2];
static double boba1 (double d) {
   a[0] = 0.0;
   return d;
}
double boba2 (double x, double y) {
   return sqrt(x*x + y*y) + boba1(x);
}
```

   (a) Suponha que compilemos esse arquivo com `gcc -c arq.c`, gerando o arquivo objeto `arq.o`. Liste todos os símbolos exportados (definidos pelo módulo) e importados (esperados de outros módulos) no arquivo `arq.o`, ou seja, o que apareceria como D (dados) ou T (texto, ou código), na primeira categoria, e, na segunda categoria, o que apareceria como U (undefined), na listagem do `nm`.

   (b) Imagine agora que ligamos esse arquivo com outro, `arqm.c`, para gerar um executável, sendo que em `arqm.c` existe a seguinte declaração:

```
float a[2] = {1.0, 2.0};
```

   Como serão os valores do array `a` depois de uma chamada a `boba2`, supondo que nenhum outro ponto do código altere esse array?

   (c) No arquivo `math.h` existe a declaração:

```
double sqrt(double x);
```

   O que aconteceria com nosso programa se omitíssemos o `include` e incluíssemos diretamente a declaração abaixo?

```
int sqrt(double x);
```