PUC-Rio – Software Básico – INF1018
Prova 2 – 27/06/2013 – Turma 3WB

1. (3,0 pontos) Traduza a função `foo` abaixo para assembly IA-32 (o assembly visto em sala), utilizando as regras usuais de alinhamento, uso de registradores, passagem de parâmetros e retorno de resultados em C. **Comente o seu código!**
(Não se preocupe se você não entender o que a função faz, apenas traduza-a literalmente).

```
double boo(double d);

int foo(float *pf, double *pd, int n) {
  double acc = 0.0;
  while (n--) {
    acc += boo(*pf + *pd);
    pf++; pd++;
  }
  return acc;
}
```

2. (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 {
  float f;
  double d;
  char c;
} x = {-2.5, 1.0, (-1) << 2};

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

Supondo que x seja armazenado no endereço de memória 0x804a014, mostre o que o programa irá imprimir quando executado. 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!)

3. (2,0 pontos) Suponha que os arquivos abaixo sejam compilados em separado, gerando os arquivos objeto `ioaux.o` e `copia.o`:

   - arquivo `ioaux.c`:

```
#include <fcntl.h>
extern int tamBufferAberto;

int myopen (const char *pathname) {
  return open(pathname, O_RDONLY);
}
```

```
int myread(int fd, void *buf) {
  return read(fd, buf, tamBufferAberto);
}
int mywrite(const void *buf, int count) {
  return write(STDOUT_FILENO, buf, count);
}
int myclose(int fd) {
  return close(fd);
}
```

- arquivo `copia.c`:

```
#include <stdio.h>
int myopen (const char *pathname);
int myread(int fd, void *buf);
int mywrite(const void *buf, int count);
int myclose(int fd);

int tamBufferAberto = 1024;
static char buf[1024];

int main (int argc, char** argv) {
  int arq, lidos;
  arq = myopen (argv[1]);
  if (arq<0) { printf("erro na abertura de arquivo %s\n", argv[1]); return 1;}
  while ((lidos = myread (arq, buf)) > 0)
    mywrite (buf, lidos);
  myclose (arq);
  return 0;
}
```

Liste, para cada um dos arquivos objeto gerados, quais símbolos apareceriam como **D**, **T** e **U** na saída do programa `nm`, considerando que, nesta saída, "D", "T" e "U" tem o significado a seguir:

- D - símbolo na área de dados, exportado
- T - símbolo na área de código, exportado
- U - símbolo indefinido

4. (1,0 ponto) Considerando programa abaixo, que utiliza a biblioteca de *co-rotinas assimétricas* que vimos no curso,

```
#include <stdio.h>
#include "corotinas.h"

/* corpo da co-rotina */
int coroFunc(int v) {
  while (v > 0) {
    printf("coro = %d\n",v);

    /* suspende a co-rotina */
    v = coro_yield(v+1);
    /* voltou */
  }
  /* termina a co-rotina */
  return;
}
```

```
/* programa principal */
int main(void) {
  int v = 0;

  /* cria uma co-rotina */
  CoroDescrP co = coro_create("co", coroFunc);

  while (v < 4) {
    /* (re)ativa a co-rotina */
    v = coro_resume(co, v+1);
    /* voltou */
    printf("main = %d\n".v);
  }
  /* avisa à co-rotina para terminar */
  coro_resume(co, -1);
  coro_free(co);
  return 0;
}
```

mostre o que o programa irá imprimir quando executado.

5. (1,5 pontos) Dada a seguinte função na linguagem "limitada" do segundo trabalho:

```
v0 = $1 + p0
ret v0
```

Escreva o código *assembly* (**não o código de máquina**) da função que seria criada pelo seu procedimento *compila*. **Esse código deve seguir o mesmo padrão utilizado no trabalho!**