Laboratório 22
Código de Máquina

1. Crie os dois arquivos abaixo

- calc.c

```
int add(int x);

int calc(int x) {
  return add(x);
}
```

- main-add.c

```
int add(int x) { return x+1; }

int calc(int x);

int main() {
   calc(10);
   return 0;
}
```

Gere os arquivos objetos e depois o executável com os comandos:

```
gcc -c -Wall calc.c
gcc -c -Wall main-add.c
gcc -o main-add  calc.o main-add.o
```

Utilize a ferramenta *objdump* (com opção -d) para decompilar *calc.o* e *main*.

Por que o opcode da instrução call (“e8”) na função calc() de *calc.o* é seguido de zeros e na função calc() do executável não? O que é esse valor que segue a instrução?

2. Crie os arquivos C abaixo:

- add.c

```
int add(int x) {
   return x+1;
}
```

- opcode.c

```
#include <stdio.h>

typedef int (*FuncPtr)(int x);

unsigned char codigo[] = {
  // Preencher com 0x??, 0x??, ...
};

int main()
{
  int i;
  FuncPtr f = (FuncPtr)codigo;

  i = f(10);
  printf("%d\n", i);

  return 0;
}
```

a) Gere o arquivo objeto de *add.c*:

```
gcc -Wall -c add.c
```

b) Edite o arquivo *opcode.c* e preencha o conteúdo do vetor *codigo* com o código de máquina do arquivo *add.o*, ou seja, faça a decompilação com *objdump* do objeto e digite cada byte das instruções como valores do vetor. Note que *objdump* mostra os hexadecimal não precedido por "0x", mas em C deve-se colocar esse prefixo, por exemplo: 0x55, 0x48, ...

c) Compile somente programa *opcode.c* com o seguinte comando e depois execute:

```
gcc -Wall -Wa,--execstack -o opcode opcode.c
```

Funcionou? Qual foi a saída? Por quê?

d) Compile somente programa com o seguinte comando e depois execute:

```
gcc -Wall -o opcode opcode.c
```

Funcionou? Qual foi a saída? Por quê?

3. Crie o arquivo C abaixo:

- displace.c

```
#include <stdio.h>

typedef int (*FuncPtr)(int);

unsigned char codigo[] = {
    // preencher com 0x??, 0x??, ...
};

int sum(int v) { return v+2; }

int mul(int v) { return 2*v; }


int main()
{
   int i;
   FuncPtr f;
   //---------------------------------------

   // Deslocamento de "sum" aqui

   //---------------------------------------
   // Chama a função
   f = (FuncPtr)codigo;
   i = f(10);
   printf("%d\n", i);

   //---------------------------------------

   // Deslocamento de "mul" aqui

   //---------------------------------------
   // Chama a função
   //f = (FuncPtr)codigo;
   //i = f(10);
   //printf("%d\n", i);

   return 0;
}
```

Os 4 bytes que seguem a instrução "e8" é um inteiro (32bits) que representa o deslocamento de onde se encontra a função a ser chamada. (em *little-endian*)

Esse deslocamento é a "distância" em bytes entre o endereço da função a ser chamada e o endereço a instrução após o *call* (próxima instrução), ou seja:

displacement = address(func) - address(next instruction) ;

Isso é conhecido como deslocamento relativo. Note que o deslocamento pode ser negativo.

a) Preencha o vetor *codigo* com os bytes das instruções do arquivo objeto *calc.o* do exercício 1.

b) No código em C, calcule o deslocamento para a função “sum” e atualize o vetor *codigo* com essa posição. Compile e execute o programa. Lembre-se da *flag* de compilação “-Wa,--execstack”.

Dicas:

- Você pode usar “&sum” para pegar o endereço da função.
- Endereços de memória são de 8 bytes (64bits).

b) Agora, calcule o deslocamento para a função “mul” e atualize o vetor com essa posição. Descomente o código no final da main(), compile e execute o programa.