DCC888 – Lista 1

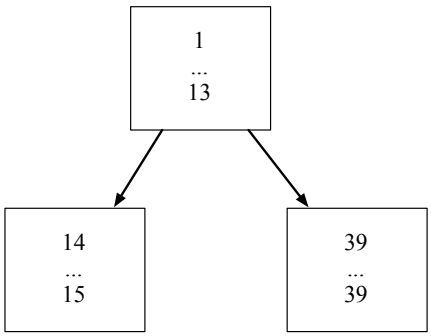
Nome: _____ Matrícula: _____

1. Este exercício refere-se à sequência de bytecodes LLVM a direita. Esta sequência foi produzida a partir do programa C visto a esquerda.

```
int main(int argc, char** argv) {
  int sum = 0;
  if (argc > 1) {
    int i = 1;
    while (i < argc) {
      char c = argv[i][0];
      if (c == 'a') {
        sum++;
      }
      i++;
    }
  }
  printf("sum = %d\n", sum);
}
```

```
1  %1 = alloca i32, align 4
2  %2 = alloca i32, align 4
3  %3 = alloca i8**, align 4
4  %sum = alloca i32, align 4
5  %i = alloca i32, align 4
6  %c = alloca i8, align 1
7  store i32 0, i32* %1
8  store i32 %argc, i32* %2, align 4
9  store i8** %argv, i8*** %3, align 4
10 store i32 0, i32* %sum, align 4
11 %4 = load i32* %2, align 4
12 %5 = icmp sgt i32 %4, 1
13 br i1 %5, label %14, label %39
14 store i32 1, i32* %i, align 4
15 br label %16
16 %8 = load i32* %i, align 4
17 %9 = load i32* %2, align 4
18 %10 = icmp slt i32 %8, %9
19 br i1 %10, label %20, label %38
20 %12 = load i32* %i, align 4
21 %13 = load i8*** %3, align 4
22 %14 = getelementptr i8** %13, i32 %12
23 %15 = load i8** %14
24 %16 = getelementptr i8* %15, i32 0
25 %17 = load i8* %16
26 store i8 %17, i8* %c, align 1
27 %18 = load i8* %c, align 1
28 %19 = sext i8 %18 to i32
29 %20 = icmp eq i32 %19, 97
30 br i1 %20, label %31, label %34
31 %22 = load i32* %sum, align 4
32 %23 = add nsw i32 %22, 1
33 store i32 %23, i32* %sum, align 4
34 br label %35
35 %25 = load i32* %i, align 4
36 %26 = add nsw i32 %25, 1
37 store i32 %26, i32* %i, align 4
38 br label %16
39 br label %40
40 %29 = load i32* %sum, align 4
41 %30 = call i32 @printf(i32 %29)
42 %31 = load i32* %1
43 ret i32 %31
```

Desenhe o CFG do programa a direita. Não é necessário escrever todas as instruções em cada bloco básico. Simplesmente escreva o primeiro e o último endereços de cada instrução naquele bloco básico. Veja o exemplo abaixo:



2. A numeração de valores (do inglês *Value Numbering*) é um dos vários métodos que compiladores usam para determinar que duas computações são equivalentes. Dessa forma, o compilador pode eliminar uma delas. A numeração de valores associa um valor simbólico a cada computação, mas sem interpretar a operação que é feita pela computação. Fazendo-se essa associação de forma sistemática, pode-se garantir que computações com o mesmo valor produzem os mesmos resultados. Esse exercício refere-se à numeração de valores.

   (a) Mostre que a propagação de constantes e a numeração de valores produzem o mesmo código otimizado para o programa abaixo:

```
i = 1
j = i + 1
k = i
l = k + 1
```

   (b) Mostre que a numeração de valores é mais forte que a propagação de constantes para o programa abaixo:

```
i = read()
j = i + 1
k = i
l = k + 1
```

   (c) Mostre o resultado da propagação de valores para o programa abaixo:

```
a = i + 1
b = 1 + i
i = j
if i + 1 goto L1
c = i + 1
```

   (d) Mostre o resultado da propagação de valores para o programa abaixo:

```
a = x ∨ y
b = x ∨ y
t1 = !z
if t1 goto L1
x = !z
c = x & y
if c goto L2
d = x & y
```