MC921 – Projeto e Construção de Compiladores

Victor Akira Hassuda Silva – 106976

Leandro Rebes Camargo – 150960

# Introdução

Este laboratório tem por objetivo verificar as otimizações proporcionadas pelos compiladores, que neste caso é o GCC. Foram compilados 4 códigos diferentes com e sem otimização e após isso foram verificados os disassemblies gerados dos executáveis, afim de compará-los entre si.

# Compilação

Foram compilados os códigos-fonte disponibilizados utilizando-se a compilação simples e sem otimização e a compilação com a otimização O3 disponibilizada pelo GCC como segue abaixo:

gcc -O0 exemploX.c -o exemploX\_O0

gcc -O3 exemploX.c -o exemploX\_O3

Após todos os binários gerados, utilizou-se a ferramenta “objdump” do próprio Sistema linux pra disponibilizar informações sobre o arquivo binário gerado pelo GCC.

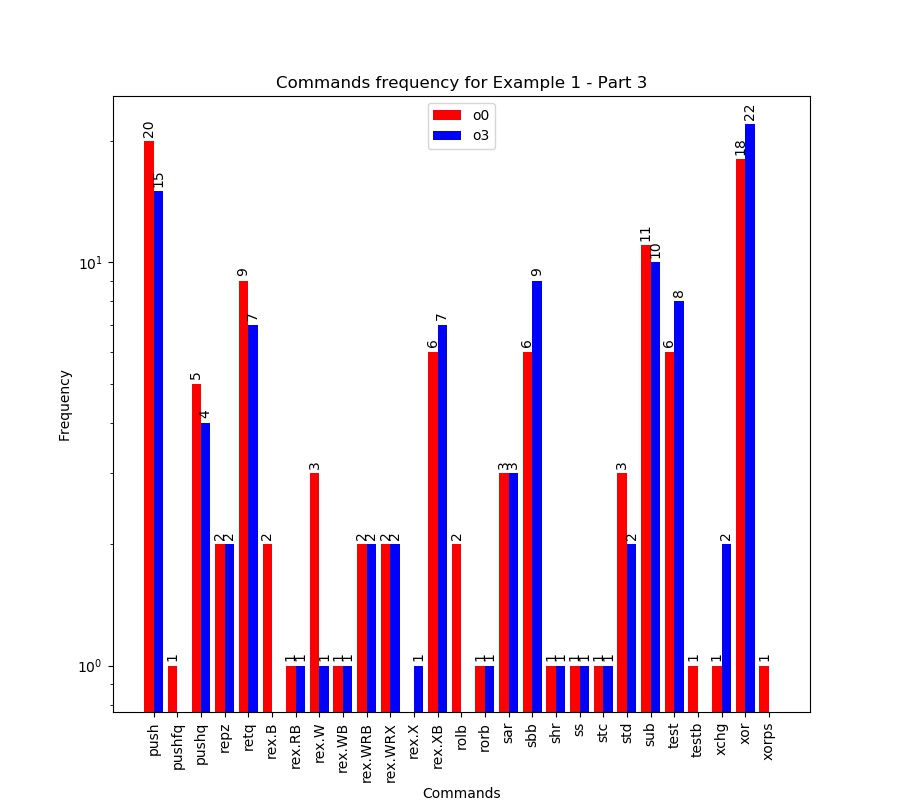
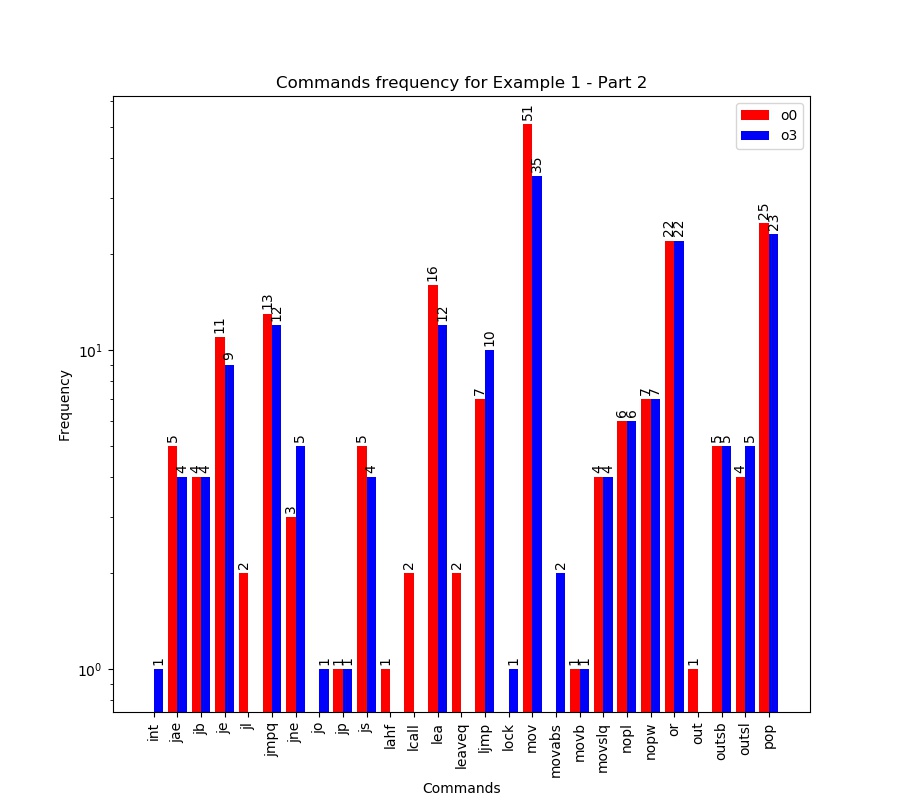
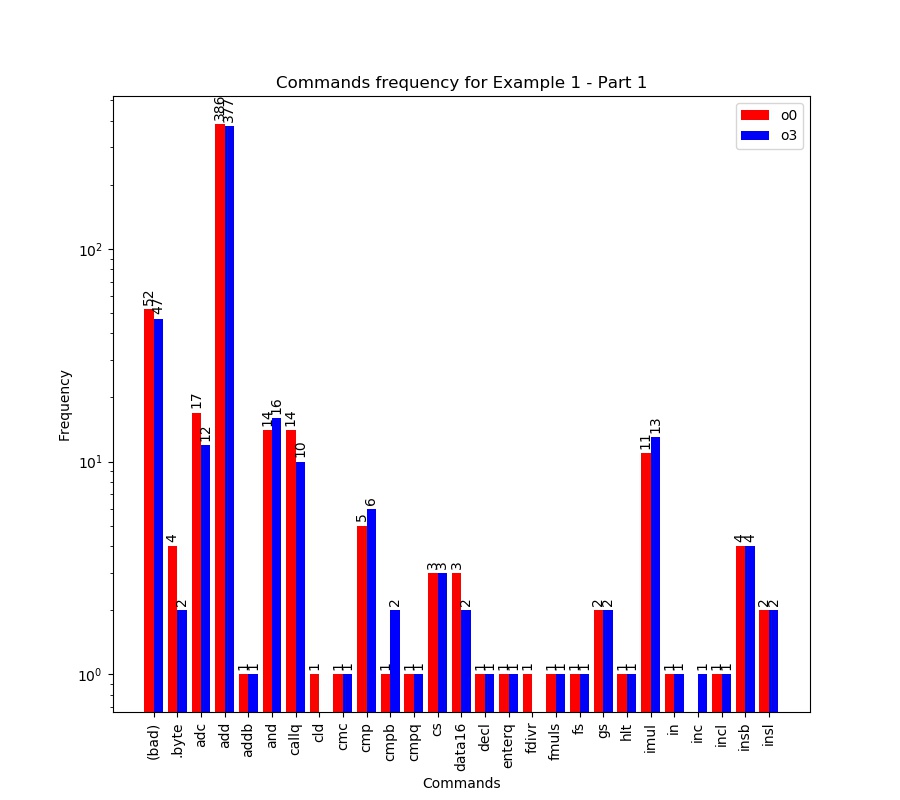
objdump -D exemploX\_OY > exemploX\_OY.dump

O arquivo dump gerado pelo “objdump” contém um código legível com instruções ASM, assim é possível analisar os dumps gerados pela compilação com e sem otimização.

# Análise

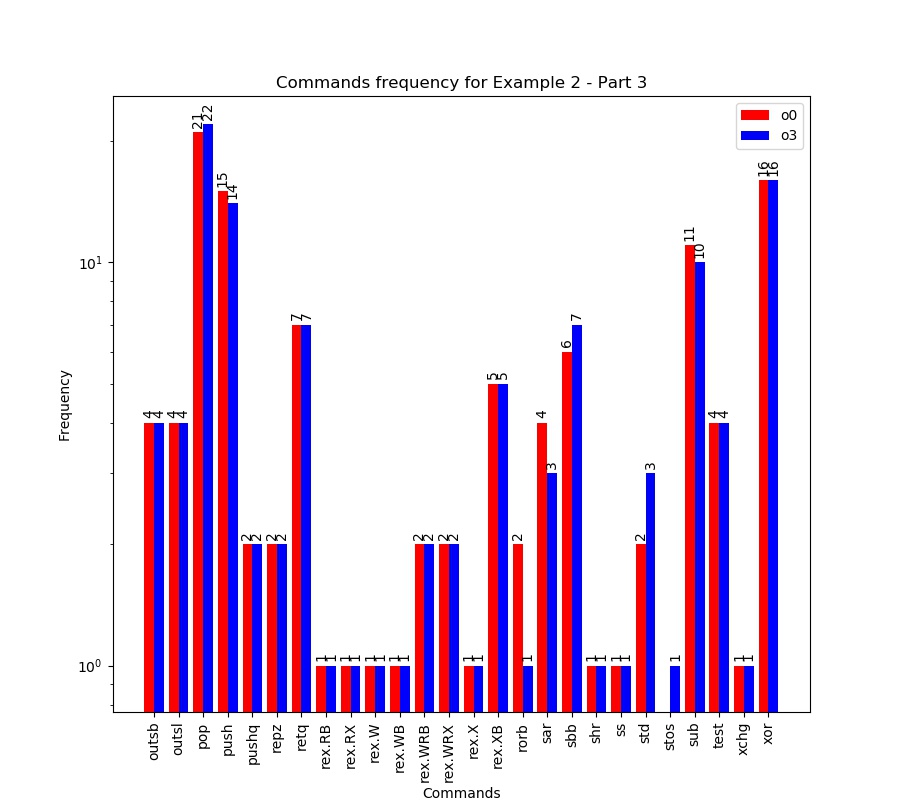
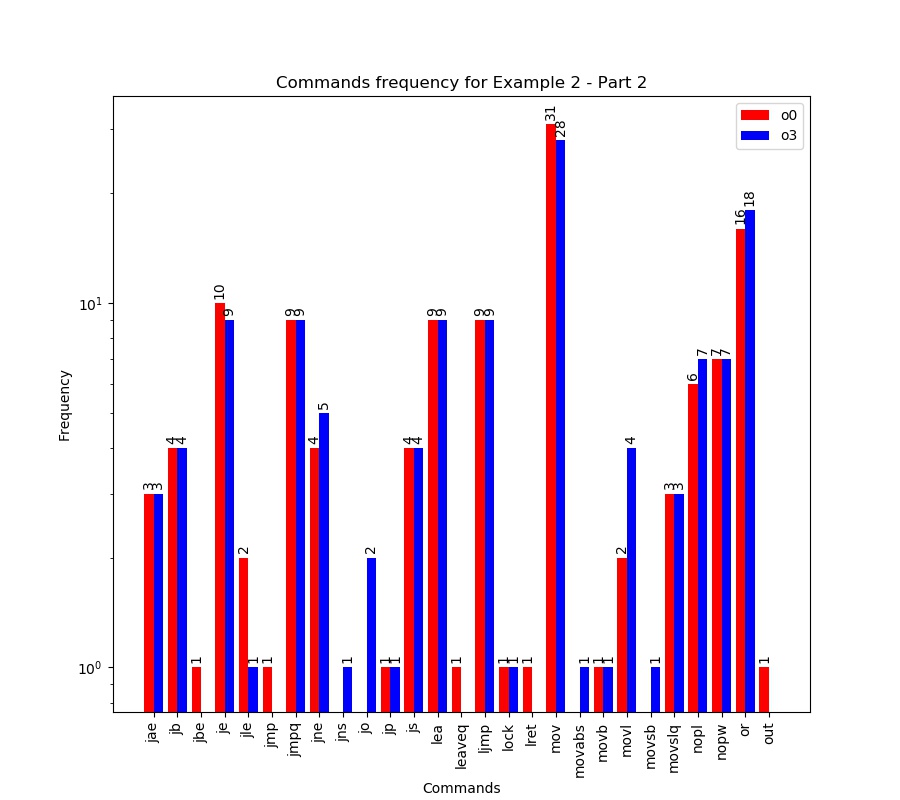
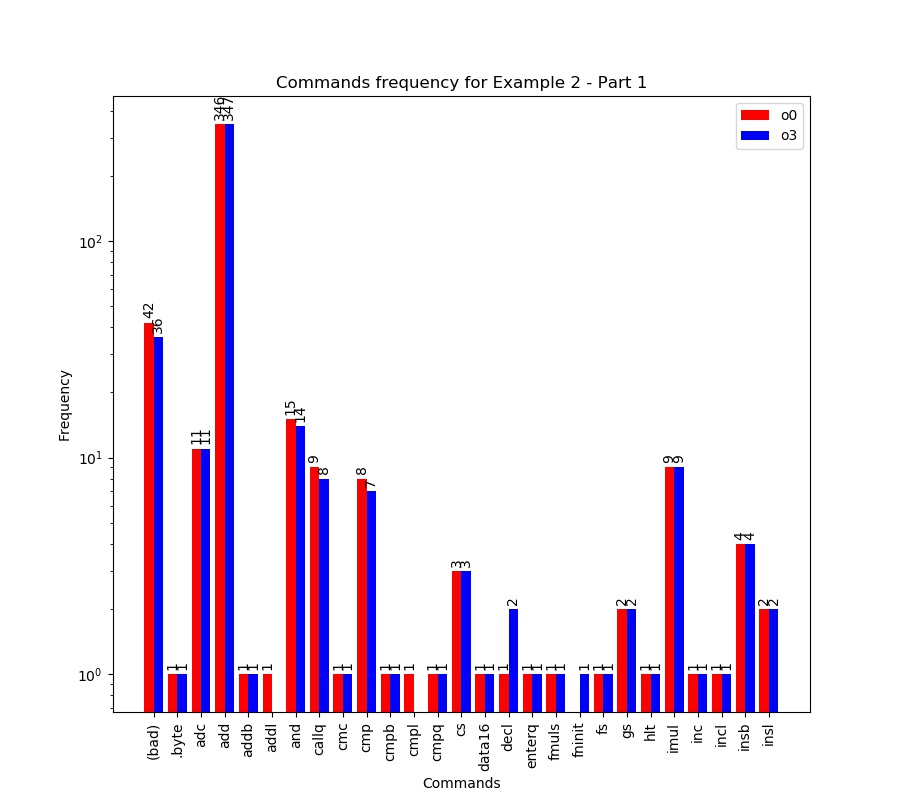
## Example1.c

Podemos verificar 2 diferenças entre o código otimizado e o código não otimizado. O primeiro possui muito menos instruções que o segundo. E a outra diferença é que a otimização eliminou as duas chamdas das funções 1 e 2, e incorporou suas ações dentro dentro da função “main”, o que torna a execução do código muito mais rápida.

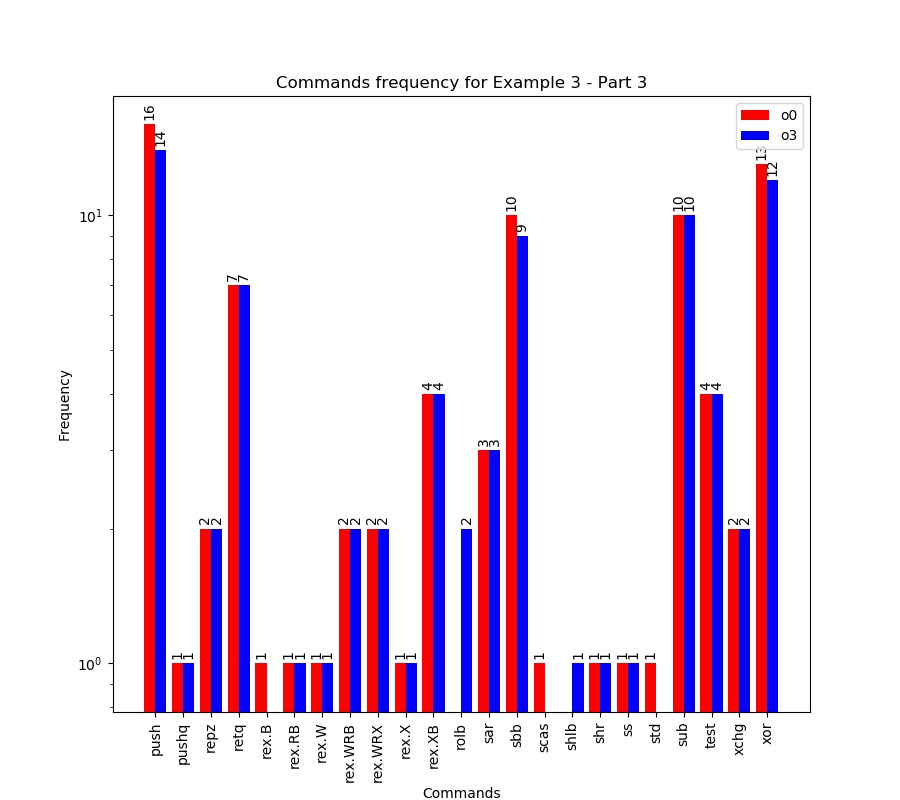
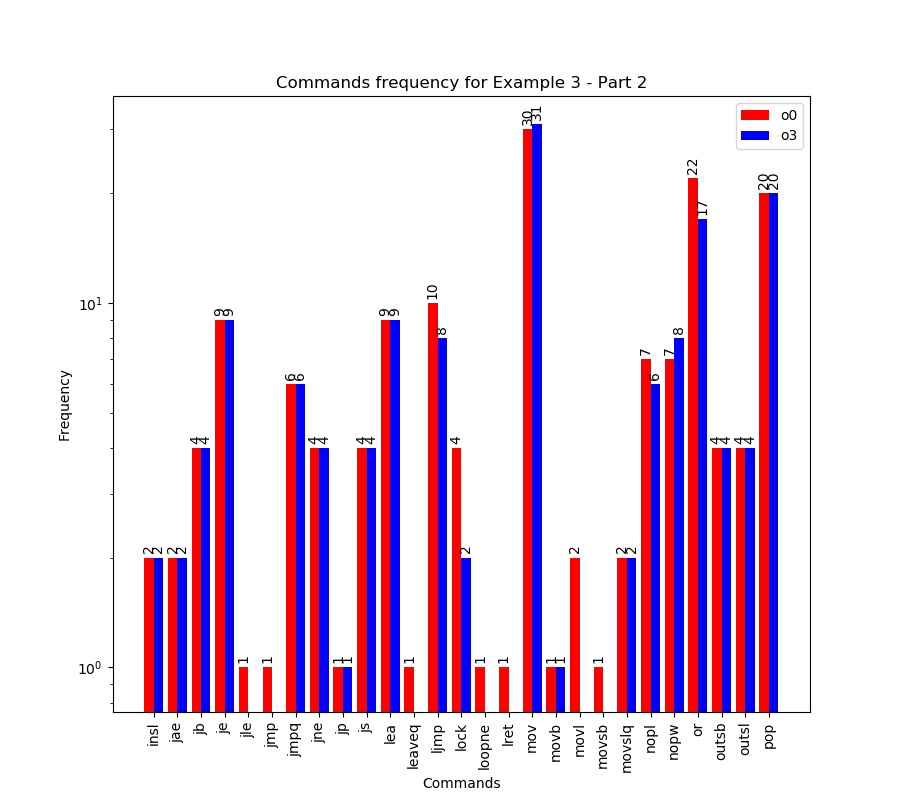
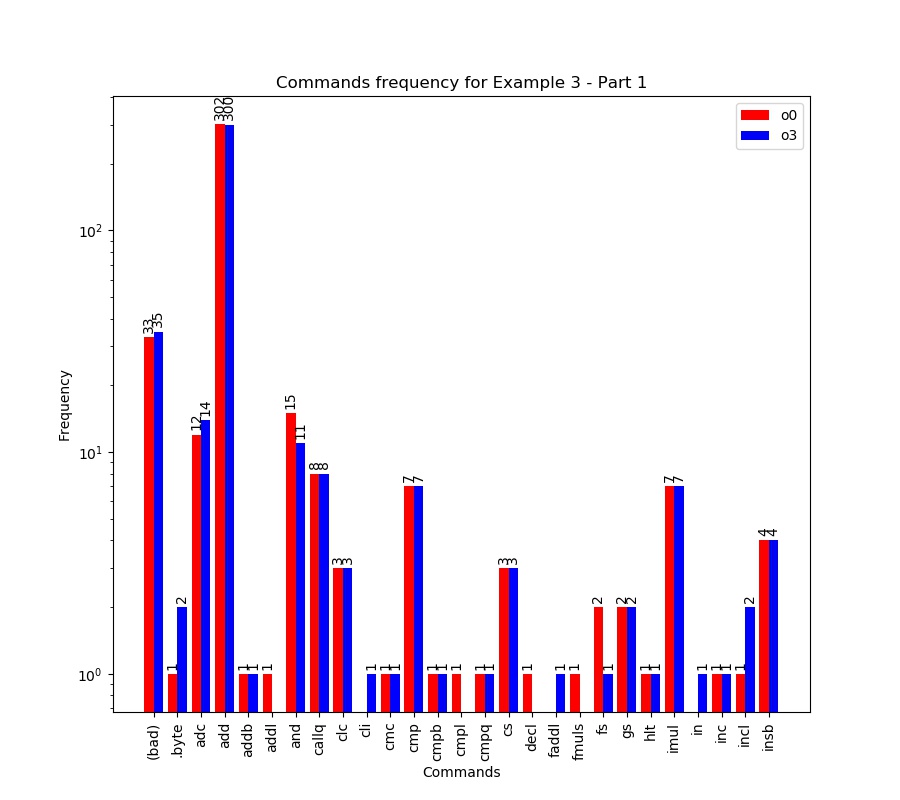
Abaixo temos um Histograma das instruções encontradas dentro do arquivo dump. E é possível notar claramente que o código compilado com otimização possui menos instruções que o código compilado sem otimização.

## Example2.c

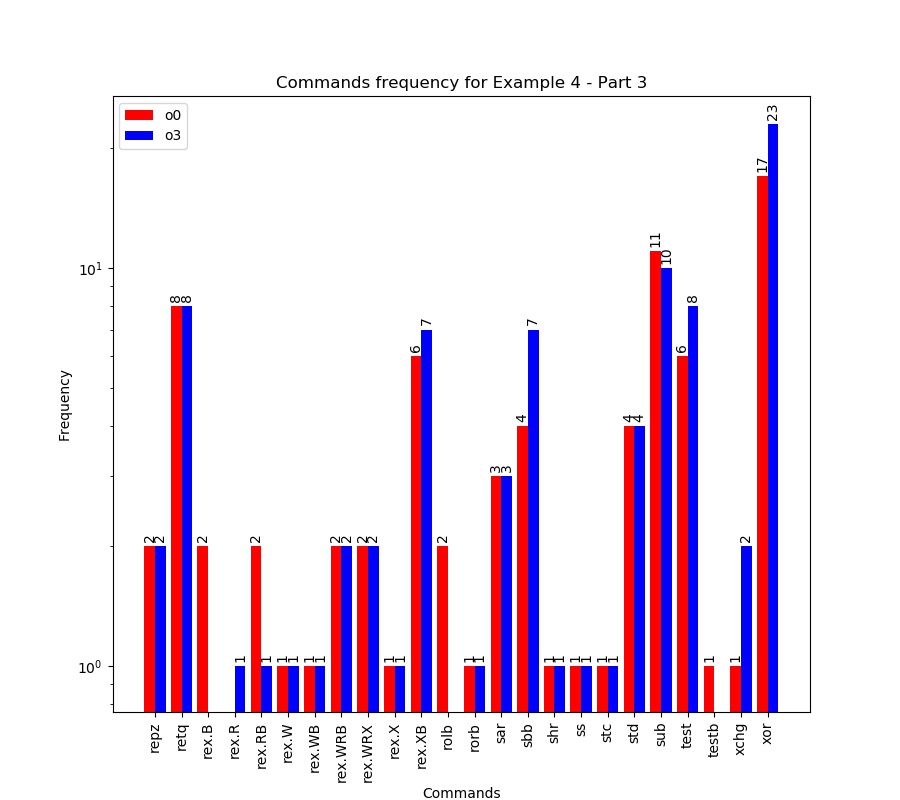
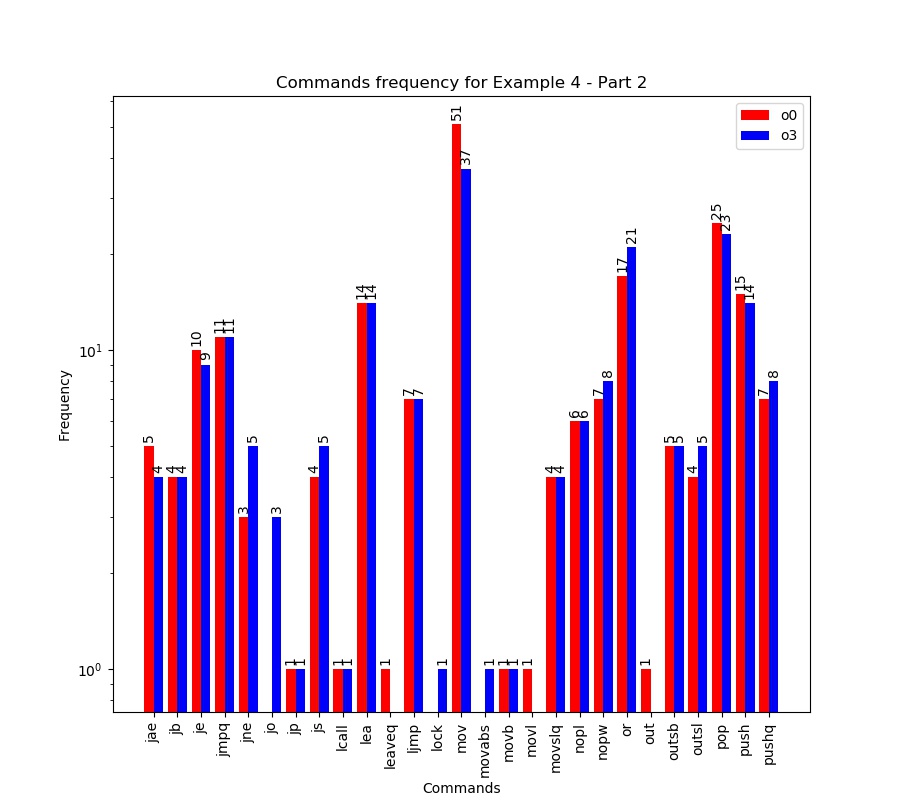
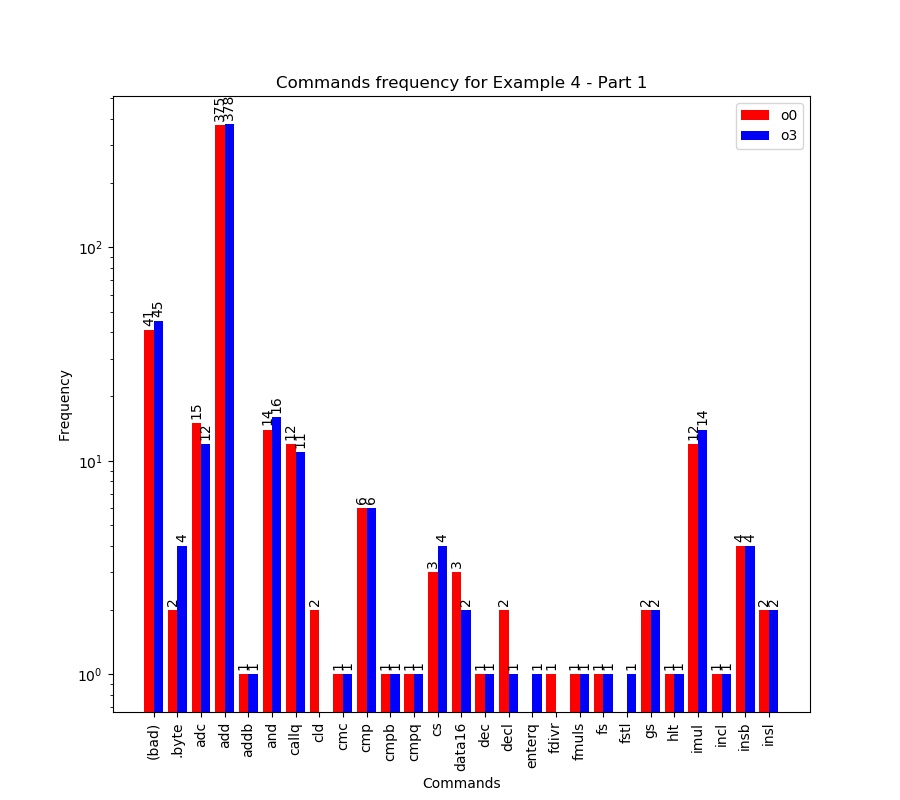
Neste código, a diferença é um pouco mais sutil, já que há pouca diferença no número de linhas de instruções. Analisando um pouco mais a fundo, pode-se observar que não há instruções de “jump” no código otimizado, e ainda mais, o laço foi removido no binário otimizado. A remoção de um laço contribui para diminuir o tempo de execução.

A seguir o Histograma do dump dos binários compilados, podemos ver que não há muita diferença na quantidade de instruções executadas.

## Example3.c

Diferentemente do código anterior que também possuia um laço, neste caso a remoção da keyword “volatile” torna a otimização do código muito mais eficiente, pois o compilador consegue saber que a variável não será alterada sem o conhecimento do programa principal.

## Example4.c

Por fim, no quarto código pode-se observar a remoção de algumas operações aritméticas e variáveis. Assim reduz-se o uso dos registradores e acessos à eles e portanto reduzindo o tempo de execução.

# Conclusões

Após analisar a seção “main” dos binários gerados pelas compilações otimizadas e não otimizadas, tivemos a possibilidade de ver como é o comportamento do GCC ao gerar os binários. Comparando todas as otimizações, a 3ª se destacou dentre as demais em número de instruções e tempo de execução.