MC921 – Projeto e Construção de Compiladores

Victor Akira Hassuda Silva – 106976

Leandro Rebes Camargo – 150960

1. 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.

2. 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
```

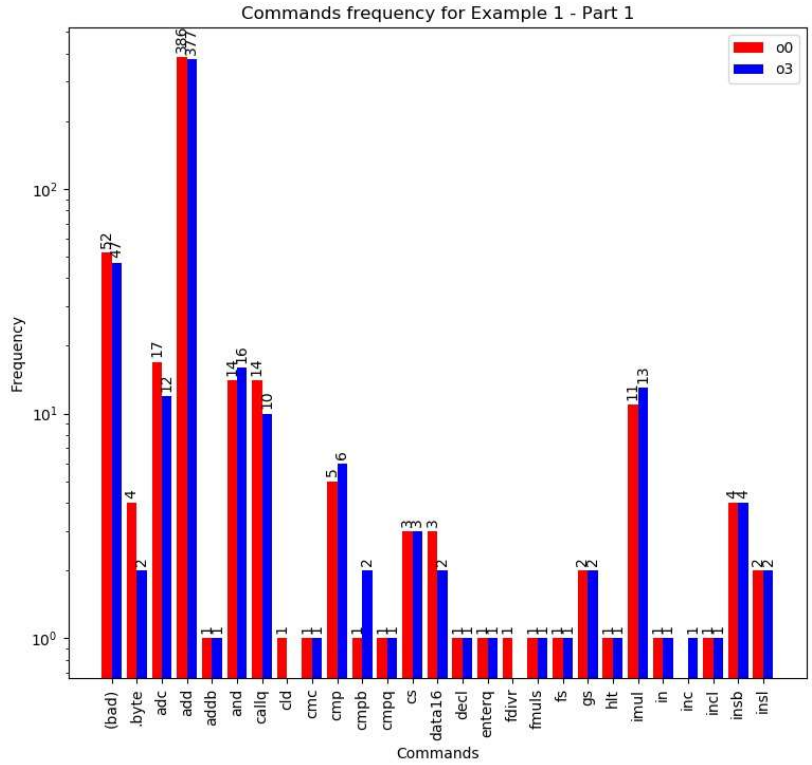
O arquivo dump gerado pelo “objdump” contém um código legível com instruções ASM, assim é possível analisar os dumps gerados pela compilação com e sem otimização.

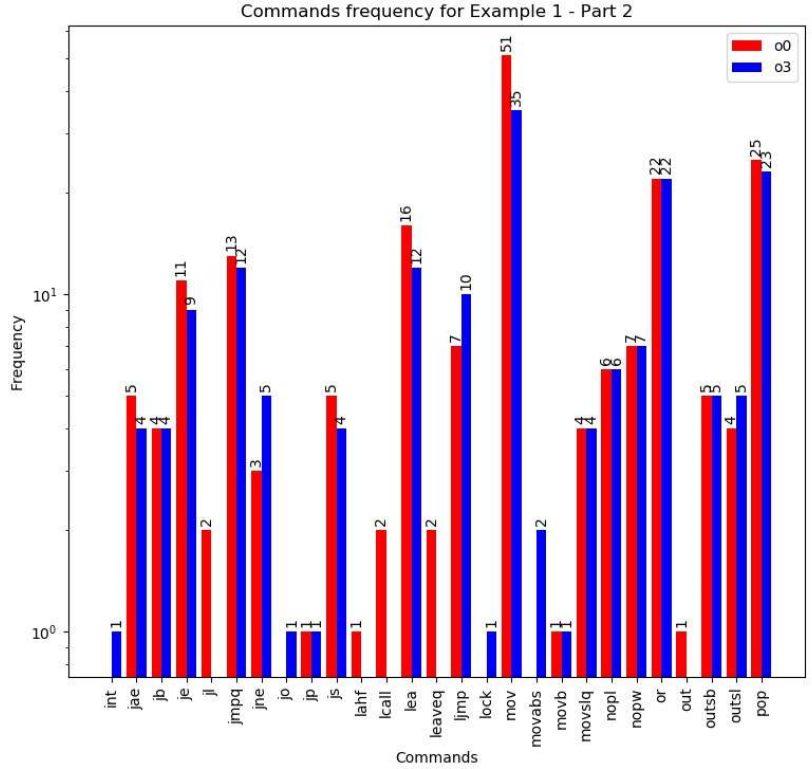
3. Análise

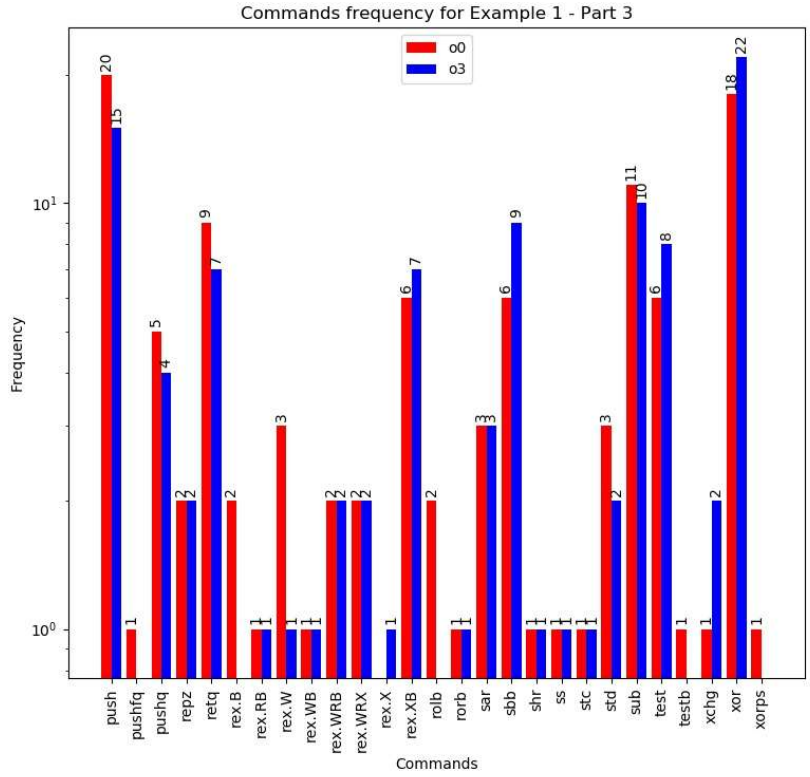
a. Example1.c

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.


Commands frequency for Example 1 - Part 1
o0
o3
Frequency
Commands
(bad)
.byte
adc
add
addb
and
callq
cld
cmc
cmp
cmpb
cmpq
cs
data16
decl
enterq
fdivr
fmuls
fs
gs
hlt
imul
in
inc
incl
insb
insl

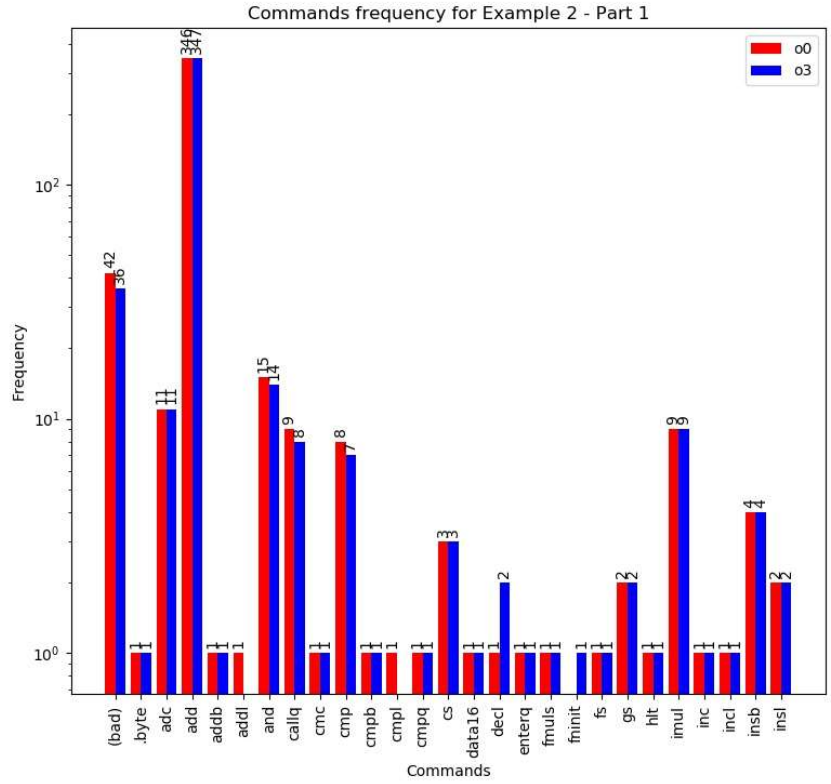

Commands frequency for Example 1 - Part 2
o0
o3
Frequency
Commands
int
jae
jb
je
jl
jmpq
jne
jo
jp
js
lahf
lcall
lea
leaveq
ljmp
lock
mov
movabs
movb
movslq
nopl
nopw
or
out
outsb
outsl
pop

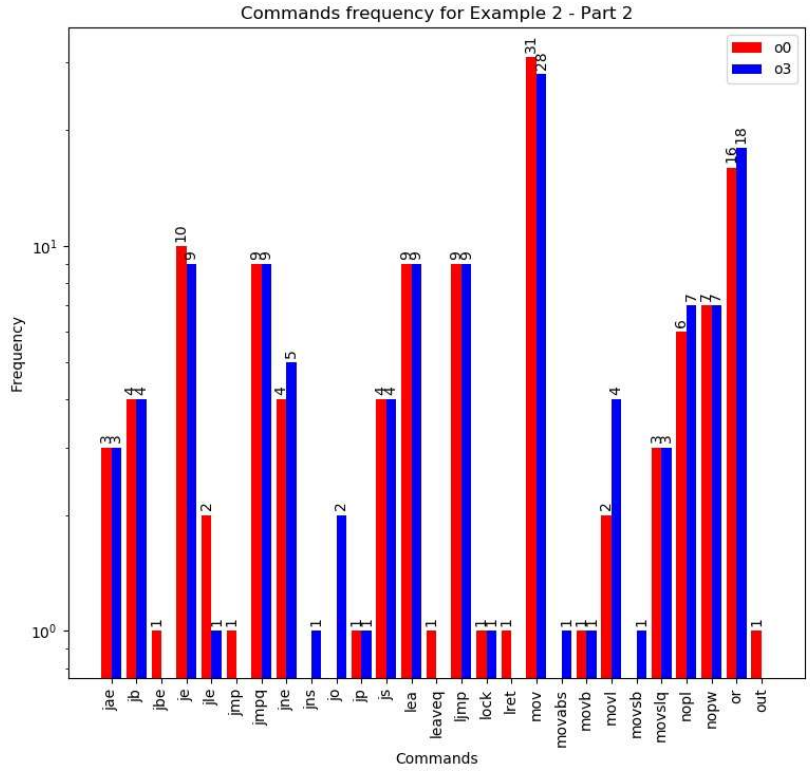

Commands frequency for Example 1 - Part 3
o0
o3
Frequency
Commands
push
pushfq
pushq
repz
retq
rex.B
rex.RB
rex.W
rex.WB
rex.WRB
rex.WRX
rex.X
rex.XB
rolb
rorb
sar
sbb
shr
ss
stc
std
sub
test
testb
xchg
xor
xorps

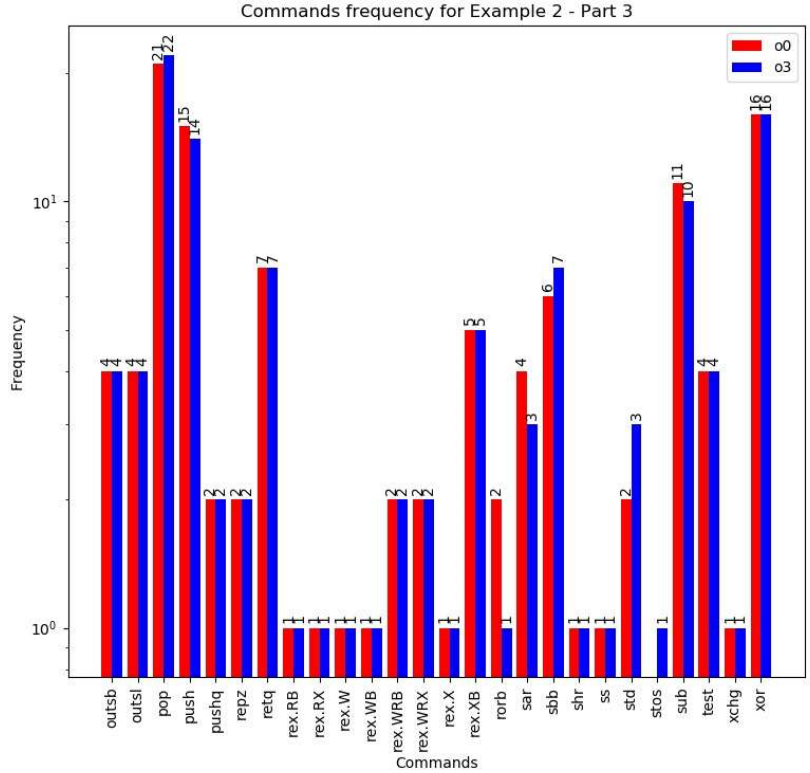
b. Example2.c

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.

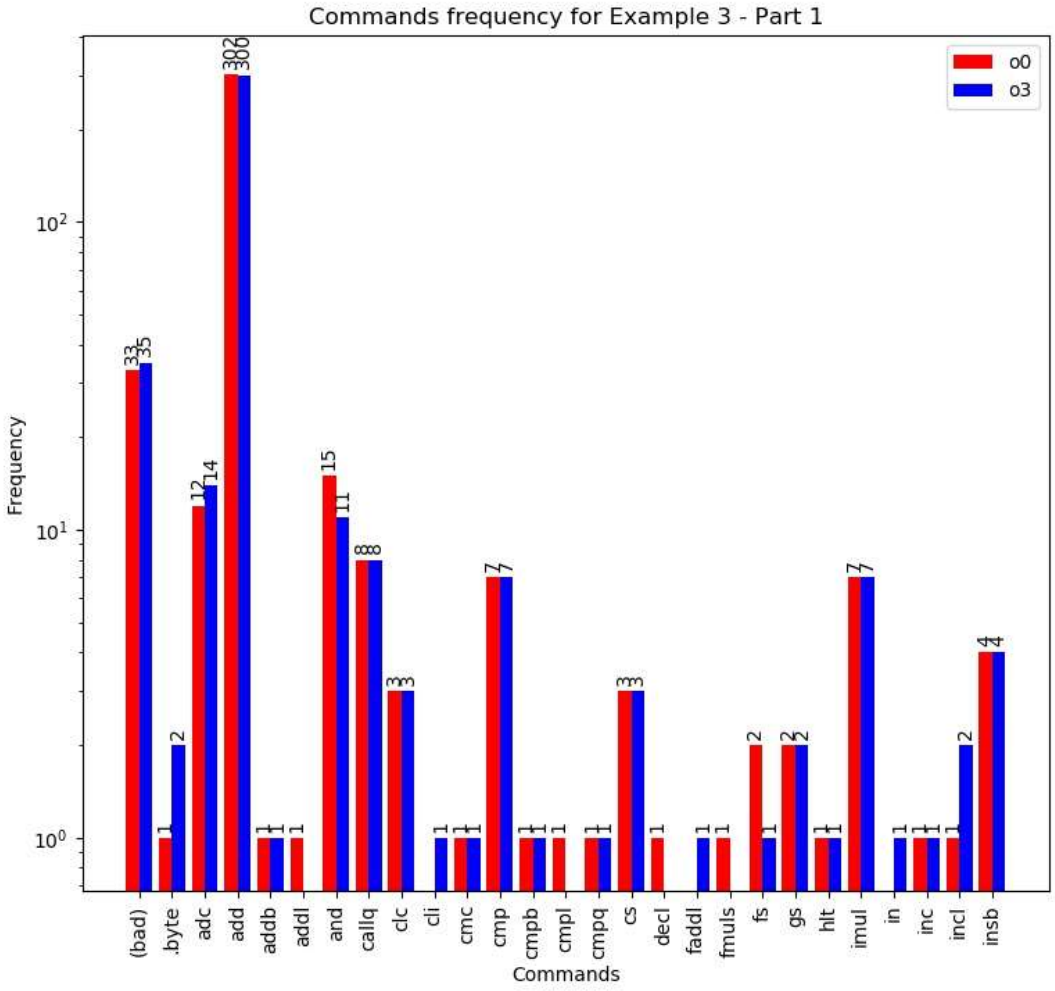


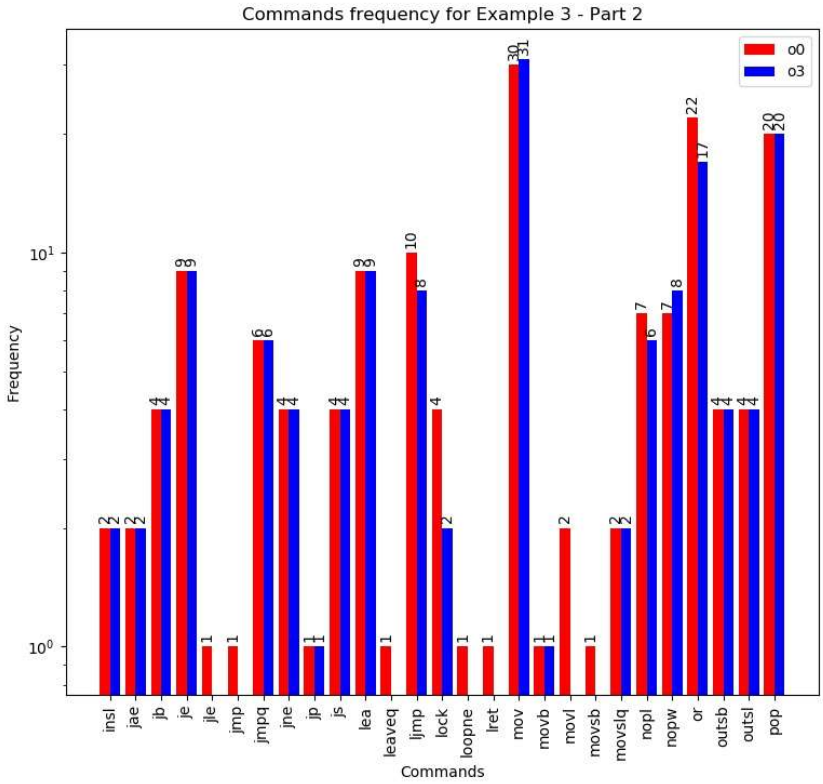

Commands frequency for Example 2 - Part 2
o0
o3
Frequency
Commands
jae
jb
jbe
je
jle
jmp
jmpq
jne
jns
jo
jp
js
lea
leaveq
ljmp
lock
lret
mov
movabs
movb
movl
movsb
movslq
nopl
nopw
or
out

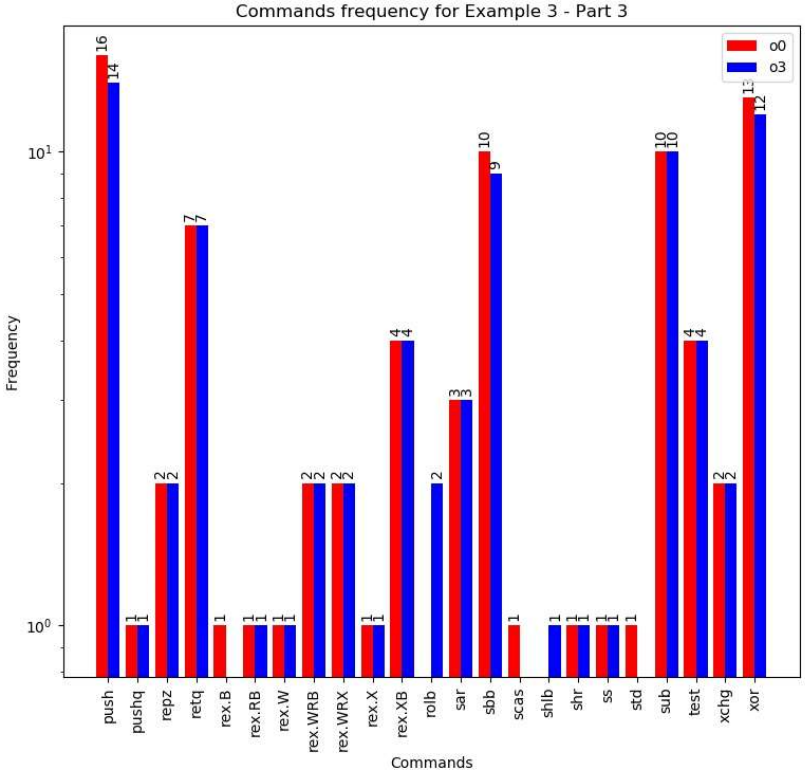

Commands frequency for Example 2 - Part 3
o0
o3
Frequency
Commands
outsb
outsl
pop
push
pushq
repz
retq
rex.RB
rex.RX
rex.W
rex.WB
rex.WRB
rex.WRX
rex.X
rex.XB
rorb
sar
sbb
shr
ss
std
stos
sub
test
xchg
xor

c. 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.

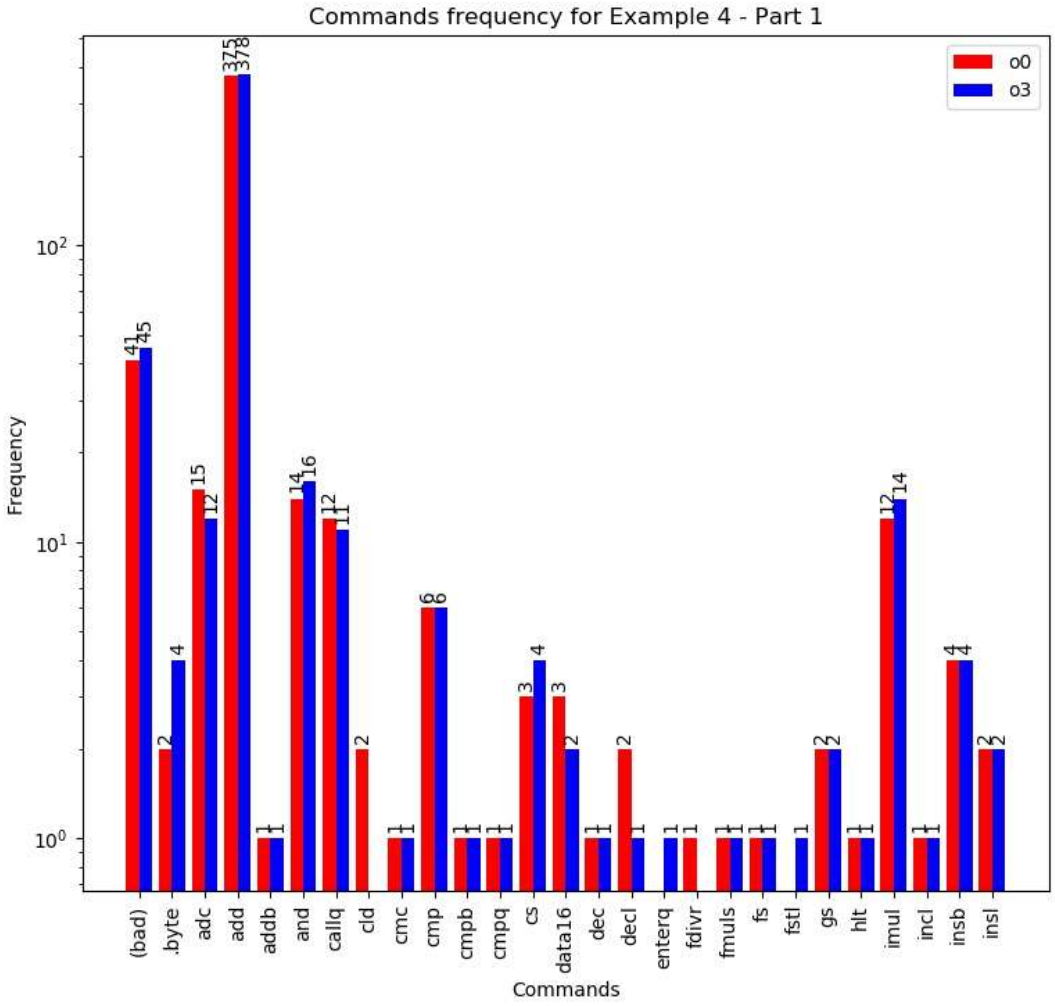


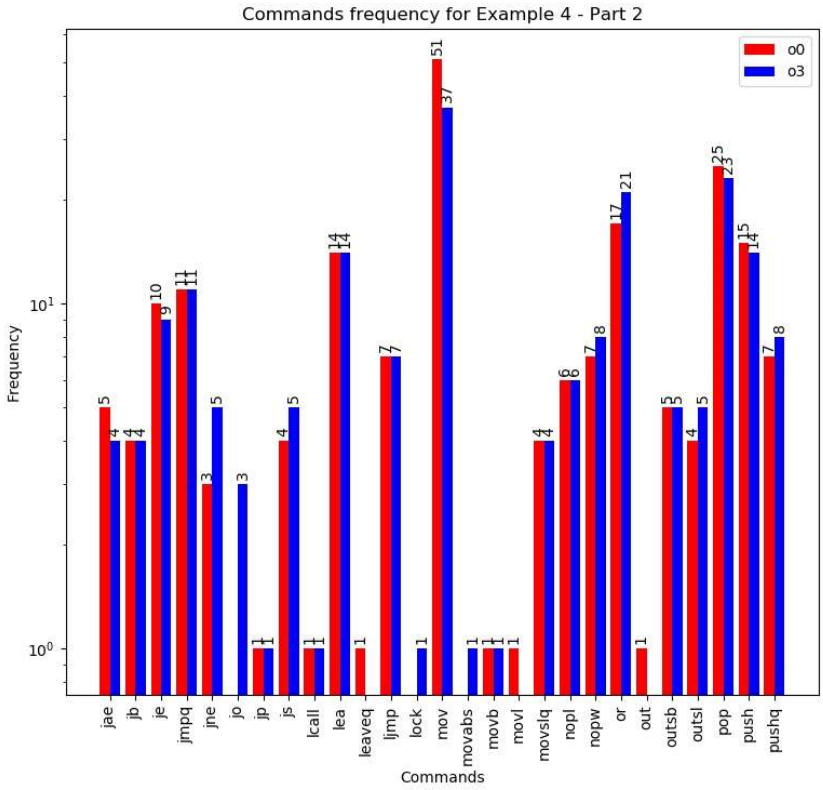

Commands frequency for Example 3 - Part 2
o0
o3
Frequency
Commands
insl
jae
jb
je
jle
jmp
jmpq
jne
jp
js
lea
leaveq
ljmp
lock
loopne
lret
mov
movb
movl
movsb
movslq
nopl
nopw
or
outsb
outsl
pop

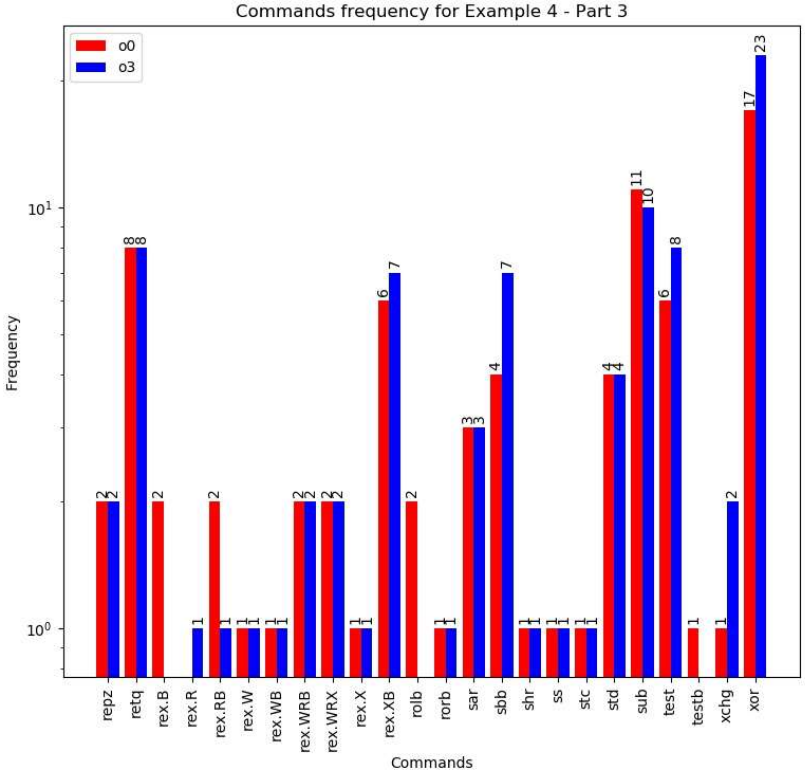

Commands frequency for Example 3 - Part 3
o0
o3
Frequency
Commands
push
pushq
repz
retq
rex.B
rex.RB
rex.W
rex.WRB
rex.WRX
rex.X
rex.XB
rolb
sar
sbb
scas
shlb
shr
ss
std
sub
test
xchg
xor

d. 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.




Commands frequency for Example 4 - Part 2
o0
o3
Frequency
10^1
10^0
jae jb je jmpq jne jo jp js lcall lea leaveq ljmp lock mov movabs movb movl movslq nopl nopw or out outsb outsl pop push pushq
Commands


Commands frequency for Example 4 - Part 3
o0
o3
Frequency
10^1
10^0
repz retq rex.B rex.R rex.RB rex.W rex.WB rex.WRB rex.WRX rex.X rex.XB rolb rorb sar sbb shr ss stc std sub test testb xchg xor
Commands

4. 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.