

АРХИТЕКТУРА РАЧУНАРА
ПРВИ ПРОЈЕКТНИ ЗАДАТАК
АСЕМБЛЕРСКИ ПРОГРАМ ЗА ОБРАДУ
ПОДАТАКА

ТЕМА: Израчунавање укупне количине простих
бројева у опсезима из специфицираног скупа
опсега

Аутор: Гордан Летић

Верзија: 1.0

Датум: 31.12.2022

Садржај:

Увод	3
Поступак и услови тестирања	3
Поређење времена извршавања	4
Поређење времена извршавања-Оптимизације gcc компајлера	8
Сликовити приказ оптимизације употребом SSE инструкцијског скупа	11
Закључак	12

УВОД

Пројектни задатак на тему асемблерске обраде података реализован је на алгоритму за израчунавање укупне количине простих бројева у опсезима из специфицираног скупа опсега. Алгоритам је реализован, како је задатком дефинисано, у асемблерском језику са стандардним инструкцијским скупом x86_64 намјењеним за линукс оперативни систем, након тога је извршена оптимизација алгоритма увођењем SSE инструкцијског скупа за паралелизацију, те на крају је реализован алгоритам у С програмском језику. Сва три рјешења су тестирана и упоређена на неколико различитих улазних података, као и различитог броја извршавања, те различитих врста компајлерских оптимизација за С програм.

ПОСТУПАК И УСЛОВИ ТЕСИРАЊА

Све реализације алгоритама су тесиране помоћу *schell* скрипте која покреће дати програм задати број пута.

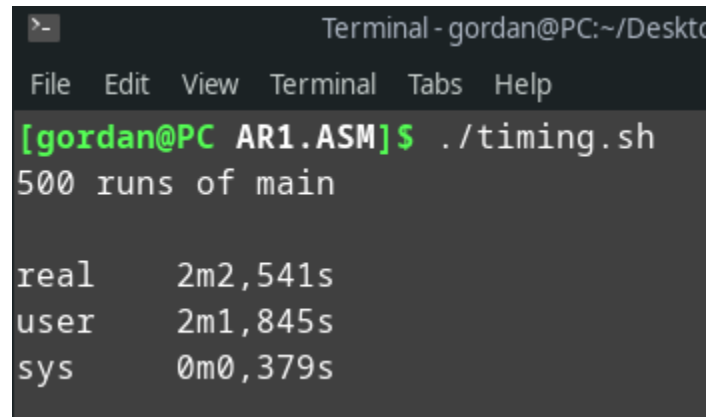
```
1  echo "500 runs of main.c"
2  time for i in {1..500}; do ./main input.bin output.bin; done
3  echo
4
5
```

Слика 1. Примјер садржаја *schell* скрипте за мјерење времена извршавања

Алгоритми су извршавани на процесору Intel i7-4510U, на оперативном систему Манјаро Linux (Arch дистрибуција). Дати процесор има 2 физичке језгре и 4 логичке (Hyper Threading), а основна фреквенција рада је 2.00GHz, а може да иде и до 3.10GHz при већем оптерећењу.

ПОРЕЂЕЊЕ ВРЕМЕНА ИЗВРШАВАЊА

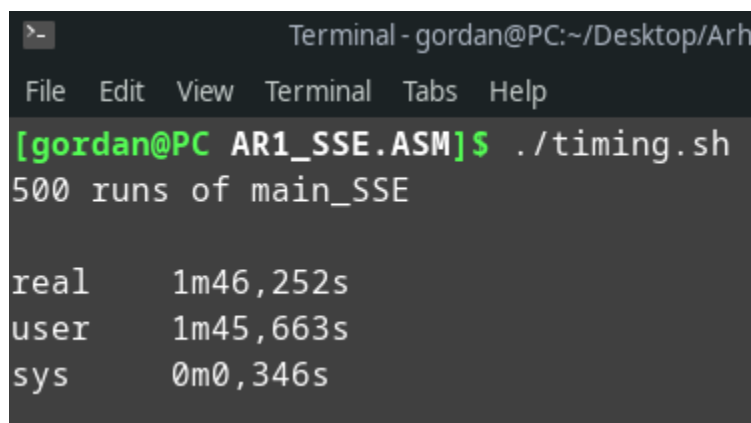
Поређење времена извршавања је представљено у виду screenshot-ова, на којима се јасно виде разлике у временима извршавања над различитим реализацијама алгоритама, као и различитих улазних података.



```
Terminal - gordan@PC:~/Desktop
File Edit View Terminal Tabs Help
[gordan@PC AR1.ASM]$ ./timing.sh
500 runs of main

real    2m2,541s
user    2m1,845s
sys     0m0,379s
```

Слика 2. X86_64: У улазном фајлу 100 опсега од 1 до 1000

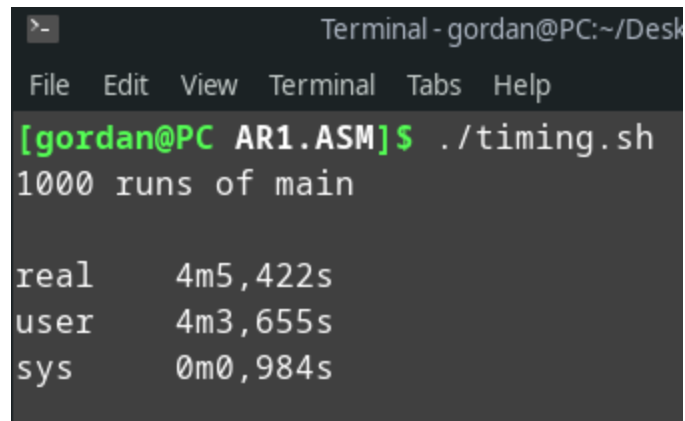


```
Terminal - gordan@PC:~/Desktop/Arh
File Edit View Terminal Tabs Help
[gordan@PC AR1_SSE.ASM]$ ./timing.sh
500 runs of main_SSE

real    1m46,252s
user    1m45,663s
sys     0m0,346s
```

Слика 3. SSE: У улазном фајлу 100 опсега од 1 до 1000

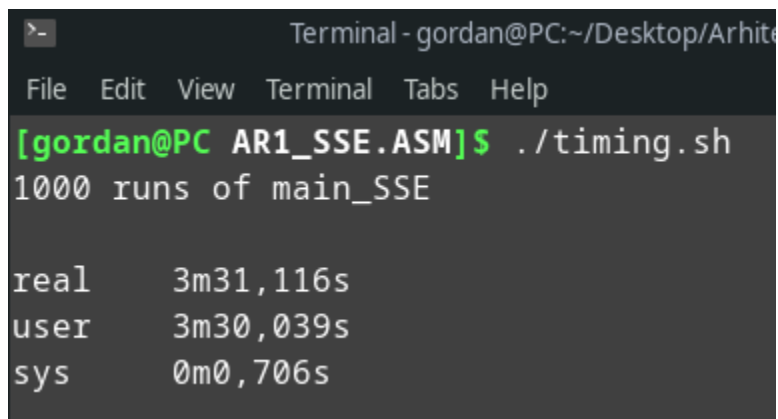
Очигледно је убрзање од приближно 16 секунди кориштењем SSE инструкцијског скупа.



```
Terminal - gordan@PC:~/Desk
File Edit View Terminal Tabs Help
[gordan@PC AR1.ASM]$ ./timing.sh
1000 runs of main

real    4m5,422s
user    4m3,655s
sys     0m0,984s
```

Слика 4. X86_64: У улазном фајлу 100 опсега од 1 до 1000

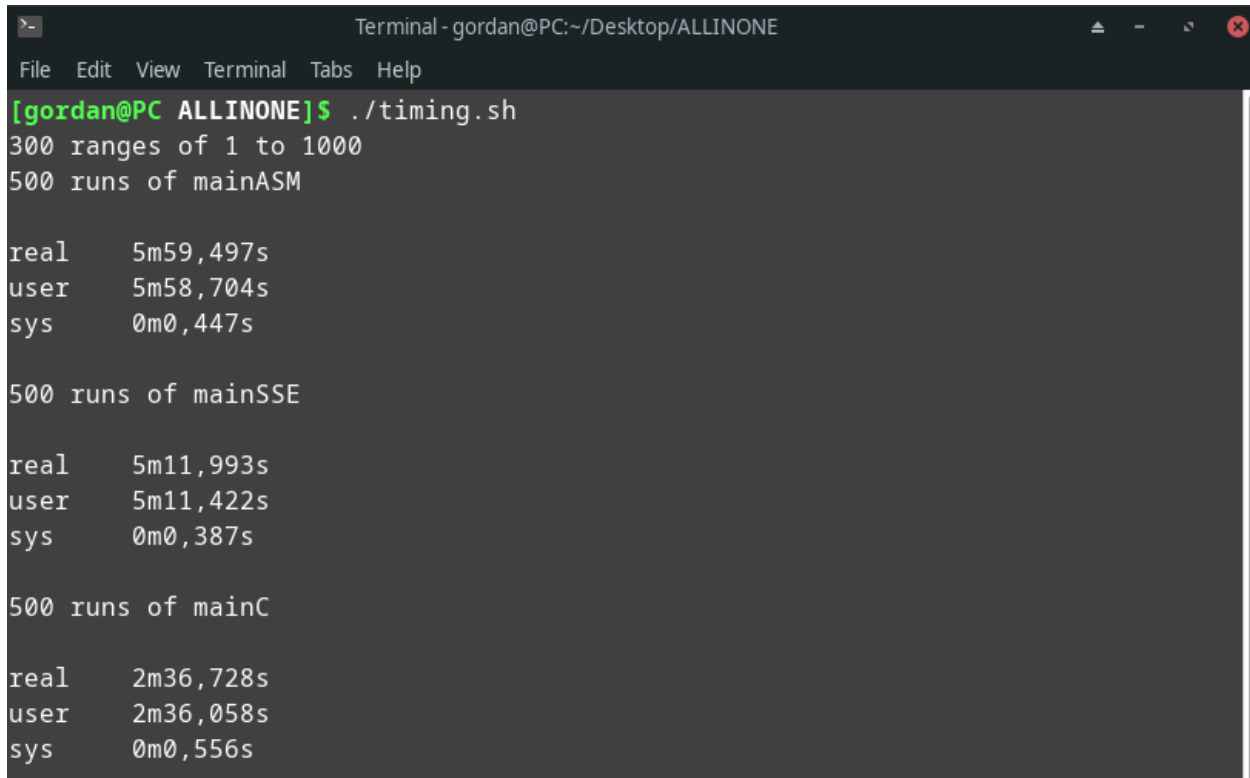


```
Terminal - gordan@PC:~/Desktop/Arhite
File Edit View Terminal Tabs Help
[gordan@PC AR1_SSE.ASM]$ ./timing.sh
1000 runs of main_SSE

real    3m31,116s
user    3m30,039s
sys     0m0,706s
```

Слика 5. SSE: У улазном фајлу 100 опсега од 1 до 1000

Очигледно је убрзање од приближно 34 секунде кориштењем SSE инструкцијског скупа, гдје видимо да је резултат пропорционалан првом мјерењу што је и очекивано.



```
Terminal - gordan@PC:~/Desktop/ALLINONE
File Edit View Terminal Tabs Help
[gordan@PC ALLINONE]$ ./timing.sh
300 ranges of 1 to 1000
500 runs of mainASM

real    5m59,497s
user    5m58,704s
sys     0m0,447s

500 runs of mainSSE

real    5m11,993s
user    5m11,422s
sys     0m0,387s

500 runs of mainC

real    2m36,728s
user    2m36,058s
sys     0m0,556s
```

Слика 6. Поређење: У улазном фајлу 300 опсега од 1 до 1000

Повећањем броја улазних елементата (опсега) на 300 уочавамо приближно убрзање од 48 секунди са кориштењем SSE инструкцијског скупа. Занимљиво да када погледамо вријеме извршавања С програма он се извршавао најбрже у свим случајевима. Када деасемблирамо С програм схватамо да је компјлер генерисао асемблерски код који је доста другачији од нашег, те је то највјероватније разлог за брже извршавање овог програма у односу на нашу имплементацију асемблерског кода.

```

Terminal - gordan@PC:~/Desktop/Arhitektura-Optimizacija/AR1
File Edit View Terminal Tabs Help
[gordan@PC AR1.C]$ objdump -d main
main: file format elf64-x86-64

Disassembly of section .init:
1000: <_init>:
1000: f3 0f 1e fa      endbr64
1004: 48 83 ec 08      sub    $0x8,%rsp
1008: 48 8b 05 c1 2f 00 00 mov    0x2fc1(%rip),%rax
100f: 48 85 c0         test   %rax,%rax
1012: 74 02           je     1016 <_init+0x16>
1014: ff d0          call   *%rax
1016: 48 83 c4 08      add    $0x8,%rsp
101a: c3             ret

Disassembly of section .plt:
1020: <fread@plt-0x10>:
1020: ff 35 ca 2f 00 00      push   0x2fca(%rip)
1026: ff 25 cc 2f 00 00      jmp     *0x2fcc(%rip)
102c: 0f 1f 40 00      nopl   0x0(%rax)

1030: <fread@plt>:
1030: ff 25 ca 2f 00 00      jmp     *0x2fca(%rip)
1036: 68 00 00 00 00      push   $0x0
103b: e9 e0 ff ff ff      jmp     1020 <_init+0x20>

1040: <fclose@plt>:
1040: ff 25 c2 2f 00 00      jmp     *0x2fc2(%rip)
1046: 68 01 00 00 00      push   $0x1
104b: e9 d0 ff ff ff      jmp     1020 <_init+0x20>

1050: <__stack_chk_fail@plt>:
1050: ff 25 ba 2f 00 00      jmp     *0x2fba(%rip)
1056: 68 02 00 00 00      push   $0x2
105b: e9 c0 ff ff ff      jmp     1020 <_init+0x20>

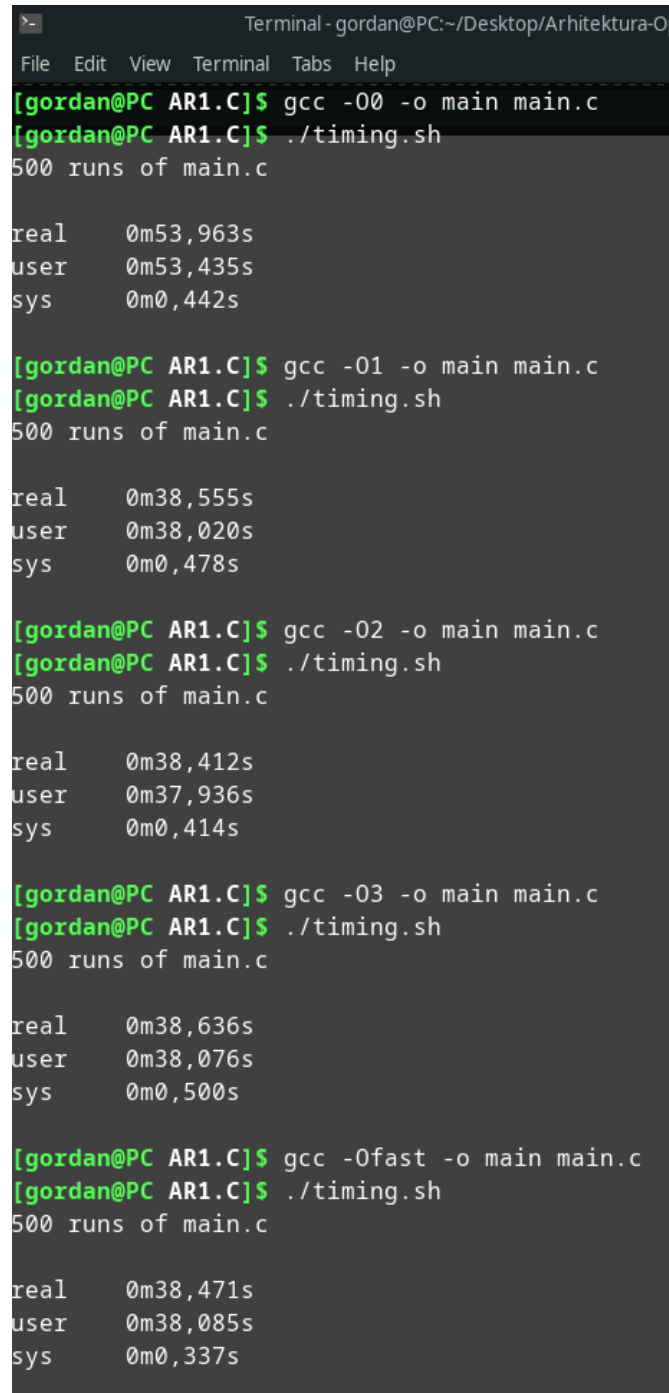
1060: <printf@plt>:
1060: ff 25 b2 2f 00 00      jmp     *0x2fb2(%rip)
1066: 68 03 00 00 00      push   $0x3

```

Слика 7. Деасемблирани С програм

ПОРЕЂЕЊЕ ВРЕМЕНА ИЗВРШАВАЊА – ОПТИМИЗАЦИЈЕ GCC КОМПАЈЛЕРА

Такође и за поређење времена извршавања при различитим компајлерским оптимизацијама, резултате извршавања ћу приложити у виду screenshot-ова на којима се јасно види које оптимизације су искориштене.



```
Terminal - gordan@PC:~/Desktop/Arhitektura-Op
File Edit View Terminal Tabs Help

[gordan@PC AR1.C]$ gcc -O0 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    0m53,963s
user    0m53,435s
sys     0m0,442s

[gordan@PC AR1.C]$ gcc -O1 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    0m38,555s
user    0m38,020s
sys     0m0,478s

[gordan@PC AR1.C]$ gcc -O2 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    0m38,412s
user    0m37,936s
sys     0m0,414s

[gordan@PC AR1.C]$ gcc -O3 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    0m38,636s
user    0m38,076s
sys     0m0,500s

[gordan@PC AR1.C]$ gcc -Ofast -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    0m38,471s
user    0m38,085s
sys     0m0,337s
```

Слика 8. С програм: У улазном фајлу 100 опсега од 1 до 1000


```
Terminal - gordan@PC:~/Desktop/Arhitektura-Op
File Edit View Terminal Tabs Help
[gordan@PC AR1.C]$ gcc -O0 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
1000 runs of main.c

real    1m47,839s
user    1m46,857s
sys     0m0,834s

[gordan@PC AR1.C]$ gcc -O1 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
1000 runs of main.c

real    1m17,136s
user    1m16,110s
sys     0m0,907s

[gordan@PC AR1.C]$ gcc -O2 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
1000 runs of main.c

real    1m16,694s
user    1m15,849s
sys     0m0,742s

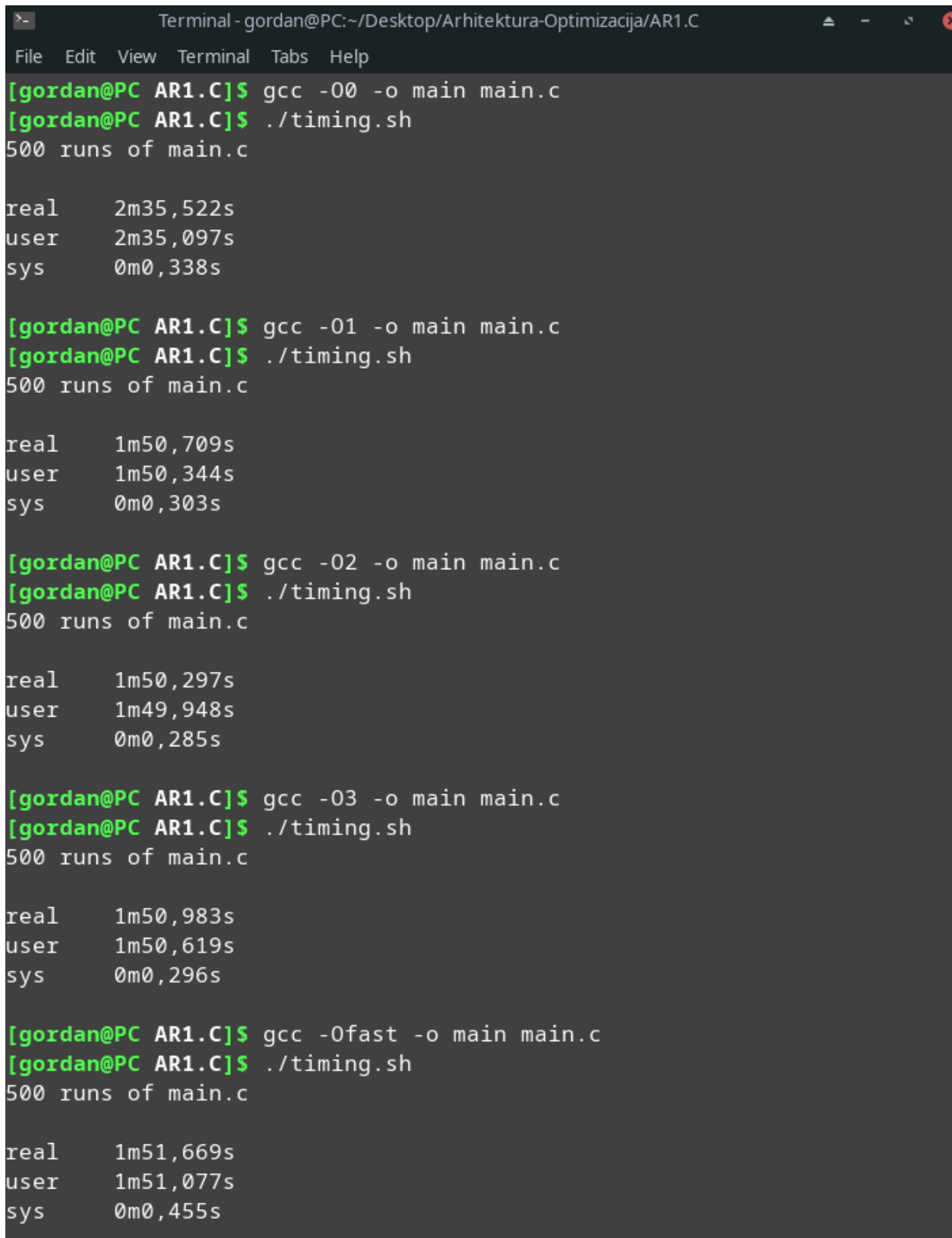
[gordan@PC AR1.C]$ gcc -O3 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
1000 runs of main.c

real    1m17,222s
user    1m16,249s
sys     0m0,846s

[gordan@PC AR1.C]$ gcc -Ofast -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
1000 runs of main.c

real    1m16,855s
user    1m15,874s
sys     0m0,873s
```

Слика 9. С програм: У улазном фајлу 100 опсега од 1 до 1000



The image shows a terminal window titled "Terminal - gordan@PC:~/Desktop/Arhitektura-Optimizacija/AR1.C". It displays the results of compiling a program named "main.c" with different optimization levels and running a timing script. The timing script runs the program 500 times and reports real, user, and system times.

```
Terminal - gordan@PC:~/Desktop/Arhitektura-Optimizacija/AR1.C
File Edit View Terminal Tabs Help

[gordan@PC AR1.C]$ gcc -O0 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    2m35,522s
user    2m35,097s
sys     0m0,338s

[gordan@PC AR1.C]$ gcc -O1 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    1m50,709s
user    1m50,344s
sys     0m0,303s

[gordan@PC AR1.C]$ gcc -O2 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    1m50,297s
user    1m49,948s
sys     0m0,285s

[gordan@PC AR1.C]$ gcc -O3 -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    1m50,983s
user    1m50,619s
sys     0m0,296s

[gordan@PC AR1.C]$ gcc -Ofast -o main main.c
[gordan@PC AR1.C]$ ./timing.sh
500 runs of main.c

real    1m51,669s
user    1m51,077s
sys     0m0,455s
```

Слика 10. С програм: У улазном фајлу 300 опсега од 1 до 1000

Као што видимо са приложених слика увођењем компајлерских оптимизација добијамо мање вријеме извршавања. С тим да након -O1 оптимизације вријеме извршавања је готово идентично. Разлог за то је вјероватно сам алгоритам у коме компајлер нема више шта да оптимизује, а за неки други програм ово не мора бити случај.

СЛИКОВИТИ ПРИКАЗ ОПТИМИЗАЦИЈЕ УПОТРЕБОМ SSE ИНСТРУКЦИЈСКОГ СКУПА

Узмимо за примјер да је број 15 број који провјеравамо да ли је прост.

Нпр. Xmm0:

15	15	15	15
----	----	----	----

Xmm1:

2	3	4	5
---	---	---	---

Xmm0 mod Xmm1:

1	0	3	0
---	---	---	---

Итерацијом кроз резултат провјеравамо да ли је остатак при дјељењу једнак нули, уколико јесте инкрементујемо бројач. Овакав поступак понављамо кроз до $n/2$ елементата. Уколико је бројач једнак нули број је прост. У овом случају слједећа итерација ће изгледати овако:

Xmm0:

15	15	15	15
----	----	----	----

Xmm1:

6	7	X	X
---	---	---	---

Xmm0 mod Xmm1:

3	1	X	X
---	---	---	---

У овом случају нас вриједности поља означених са X не занимају.

Вриједност бројача ће бити 2 што значи да број није прост.

ЗАКЉУЧАК

Након свих тестирања и упоређивања резултата долазимо до закључка да су компајлери много напредовали, те да је наш код само листа жеља којима указујемо шта желимо постићи. Док ће компајлер узети наш код пресложити и оптимизовати га на начин да омогући што боље вријеме извршавања. Овакав случај се често сусреће и у пракси, гдје поједине дистрибуције линукса (Arch) компајлирају комплетан програм на нашем хардверу како би се омогућиле и постигле што боље перформансе за циљани хардвер. Осим тога закључујемо да употребом векторског инструкцијског скупа можемо ефикасно оптимизовати извршавање нашег програма, што у суштини и сам компајлер ради у одређеним случајевима.