ECE454 Assignment 1 Report

Objectives

- Reviewing/Working with the Compilation Flags (-g, -O0, -O1, -O2, -O3, ...)
- Reviewing/Working with the Profiling Tools (time, gprof, gcov)Reviewing/Working with the Inspection Tool (objdump)

NOTE- This document is version2 of A1 with the same total marks. You can submit either solution for A1(v1) or A1(v2).

Update Note about Profiling (A1)

Profiling with gprof: Use the provided file *gprof_test.c* and the following commands in order to:

1) compile it with –gp option (enable profiling),

2) execute it to generate *gmon.out* and then

3) execute it with *gmon.out* as parameter to generate *analysis.txt* file that contains the required profiling information (flat profile and call graph, you may want to use –a or –b to suppress the information)

Profiling with gcov: Use the provided file *gcov_test1.c* and the following commands in order to:

1) Instrument the code with the test information using -ftest-coverage and -fprofile-arcs parameters and

2) run it to see the result

1.1 Profiling

Q1-The previous question has been changed to the following:

Using the Extension-Note about Profiling, go through the instructions, create profiling information using gprof and gcov, and finally display/discuss about the results. (describe the generated report for each prof and gcov options briefly). [3 mark]

A. gprof

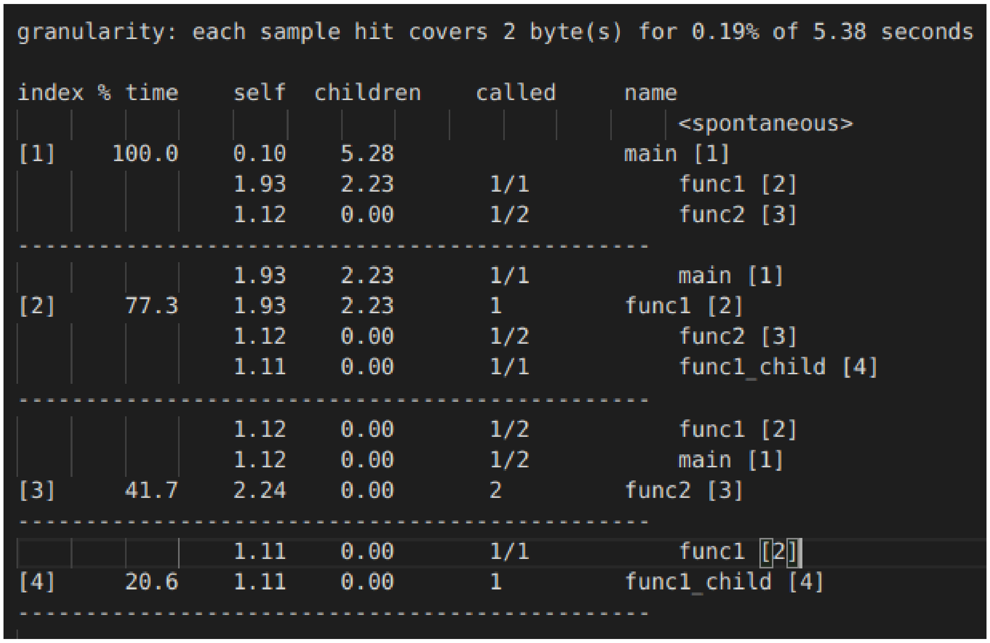
Based on the output of gprof, the hottest function (most time spent) is func 2. It is called twice in the program. The second hottest is func1. The third hottest is func1_child. The least time spent is in main.

1. gprof gprof_test gmon.out > analysis.txt

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
41.92      2.24     2.24        2     1.12     1.12  func2
36.09      4.17     1.93        1     1.93     4.16  func1
20.68      5.28     1.11        1     1.11     1.11  func1_child
1.88      5.38     0.10                             main
```

This is the default output for gprof. It gives the % of total time spent in each function. It also gives the time in seconds spent in each function. Information explaining each field in the tables is printed.

The call tree is also printed.



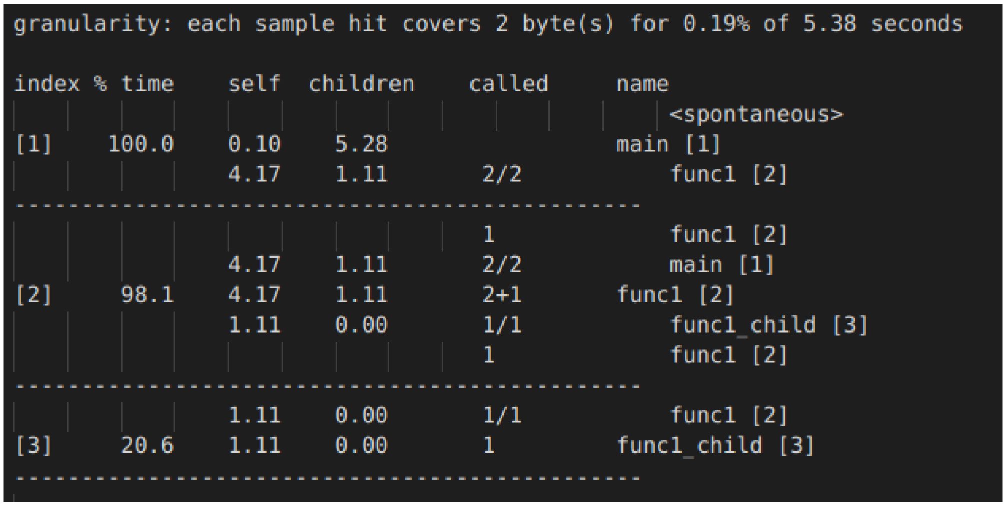
```
granularity: each sample hit covers 2 byte(s) for 0.19% of 5.38 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.10    5.28                 main [1]
                1.93    2.23       1/1           func1 [2]
                1.12    0.00       1/2           func2 [3]
-----------------------------------------------
                1.93    2.23       1/1           main [1]
[2]     77.3    1.93    2.23       1         func1 [2]
                1.12    0.00       1/2           func2 [3]
                1.11    0.00       1/1           func1_child [4]
-----------------------------------------------
                1.12    0.00       1/2           func1 [2]
                1.12    0.00       1/2           main [1]
[3]     41.7    2.24    0.00       2         func2 [3]
-----------------------------------------------
                1.11    0.00       1/1           func1 [2]
[4]     20.6    1.11    0.00       1         func1_child [4]
-----------------------------------------------
```

2. gprof –a gprof_test gmon.out > analysis.txt

```
 %   cumulative   self              self     total
time   seconds   seconds    calls  s/call   s/call  name
78.01      4.17     4.17        2    2.09     2.64  func1
20.68      5.28     1.11        1    1.11     1.11  func1_child
 1.88      5.38     0.10                            main
```

The `-a` flag suppresses the printing of statically declared functions. We can see that func2 is removed from the output. The time spent in `func2` is given to `func1` as `func1` was loaded directly before `func2` in the executable.

The corresponding call tree is also printed with `func2` omitted.

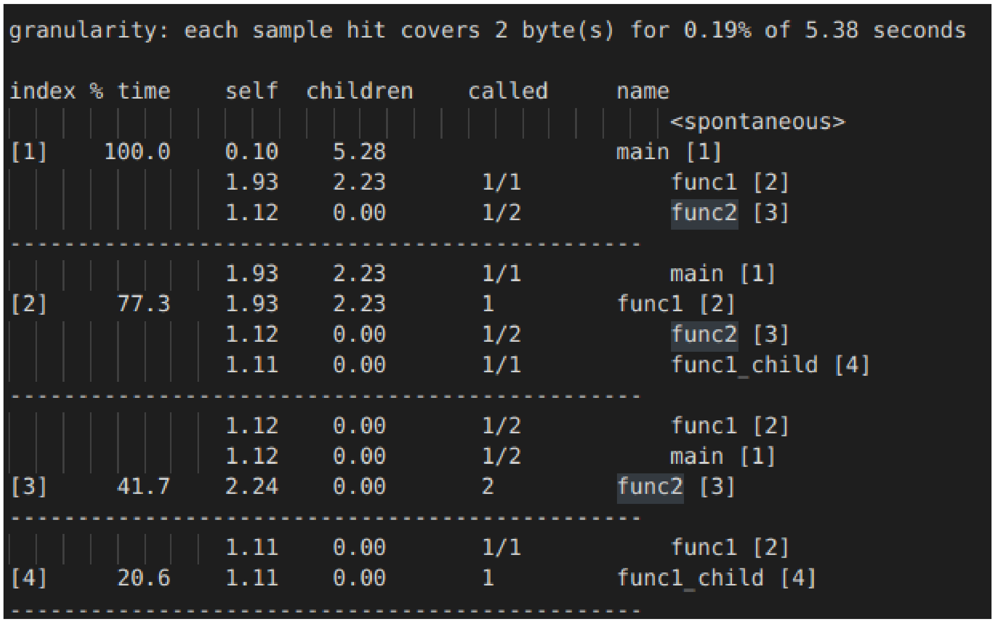


```
granularity: each sample hit covers 2 byte(s) for 0.19% of 5.38 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.10    5.28                 main [1]
                4.17    1.11       2/2           func1 [2]
-----------------------------------------------
                                   1             func1 [2]
                4.17    1.11       2/2           main [1]
[2]     98.1    4.17    1.11       2+1       func1 [2]
                1.11    0.00       1/1           func1_child [3]
                                   1             func1 [2]
-----------------------------------------------
                1.11    0.00       1/1           func1 [2]
[3]     20.6    1.11    0.00       1         func1_child [3]
-----------------------------------------------
```

3. gprof –b gprof_test gmon.out > analysis.txt

```
 %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
41.92      2.24     2.24        2    1.12     1.12  func2
36.09      4.17     1.93        1    1.93     4.16  func1
20.68      5.28     1.11        1    1.11     1.11  func1_child
 1.88      5.38     0.10                            main
```

The `-b` flag makes the output brief. With this flag, gprof removes the explanations of all the fields in the tables. Only the relevant data is printed. Its corresponding information is omitted. The call tree is also printed.



```
granularity: each sample hit covers 2 byte(s) for 0.19% of 5.38 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.10    5.28                 main [1]
                1.93    2.23       1/1           func1 [2]
                1.12    0.00       1/2           func2 [3]
-----------------------------------------------
                1.93    2.23       1/1           main [1]
[2]     77.3    1.93    2.23       1         func1 [2]
                1.12    0.00       1/2           func2 [3]
                1.11    0.00       1/1           func1_child [4]
-----------------------------------------------
                1.12    0.00       1/2           func1 [2]
                1.12    0.00       1/2           main [1]
[3]     41.7    2.24    0.00       2         func2 [3]
-----------------------------------------------
                1.11    0.00       1/1           func1 [2]
[4]     20.6    1.11    0.00       1         func1_child [4]
-----------------------------------------------
```

B. gcov

```
        -:    0:Source:gcov_test1.c
        -:    0:Graph:gcov_test1.gcno
        -:    0:Data:gcov_test1.gcda
        -:    0:Runs:1
        -:    1:// gcov_test1.c
        -:    2:#include<stdio.h>
        -:    3:
        -:    4:static void func2(void);
        -:    5:
        -:    6:void
        1:    7:func1_child(void)
        -:    8:{
        1:    9:  printf("\nInside func1_child()\n");
        1:   10:  unsigned int i = 0;
        -:   11:
```

```
4294965112:   12: for (; i < 0xfffff777; i++);
        -:   13:
        1:   14:  return;
        -:   15:}
        -:   16:
        -:   17:void
        1:   18:func1(void)
        -:   19:{
        1:   20:  printf("\nInside func1()\n");
        1:   21:  unsigned int i = 0;
        -:   22:
4294967296:   23: for (; i < 0xffffffff; i++);
        1:   24:  func1_child();
        -:   25:
        1:   26:  func2();
        1:   27:  return;
        -:   28:}
        -:   29:
        -:   30:static void
        2:   31:func2(void)
        -:   32:{
        2:   33:  printf("\nInside func2()\n");
        2:   34:  unsigned int i = 0;
        -:   35:
8589926402:   36: for (; i < 0xfffff000; i++);
        2:   37:  return;
        -:   38:}
        -:   39:
        -:   40:int
        1:   41:main(void)
        -:   42:{
        1:   43:  printf("\nInside main()\n");
        1:   44:  int i = 0;
        -:   45:
268435456:   46:  for (; i < 0xfffffff; i++);
        1:   47:  func1();
        1:   48:  func2();
        -:   49:
        1:   50:  return 0;
        -:   51:}
```

1. The first section shows the file path, source file name, graph file name, data file name, and the number of runs.
2. The second section shows the source code of the program with line numbers and the execution count for each line of code

   We can see that there are 4 loops (line 12,23,36,46) that are run several times. The rest of the code is only run once or twice. The rest of the code is only run once or twice.The rest of the code is only run once or twice.

From Q2 to Q14, measure the compilation time using the gcc command with these compilation flags at the command line: 1) -g, 2) -O2, 3) -O3, 4) –Os, then answer the following questions based on the results:

1.2 Measuring Compilation Time

Find the compilation-time with all four flags used in the previous section:

Q2-Report the 4 compilation time measurements. [1 mark]

`-g`	0.022s
`-O2`	0.032s
`-O3`	0.043s
`-Os`	0.030s

Q3-Which compilation time is the slowest and why (describe briefly)? [1 mark]

`-O3` is the slowest. Optimization level `-O3` runs the most number of optimisations. Therefore it is the slowest.

Q4-Which compilation time is the fastest and why (describe briefly)? [1 mark]

`-g` is the fastest. `-g` does not turn on any optimization. Therefore its the fastest.

Q5-Based on the results in Q1, which of gprof and gcov is faster and why (describe briefly)? [1 mark]

`gcov`	0.034s
`gprof`	0.029s

`gprof` is faster. At the compilation stage `gprof` instruments the code at the granularity of functions. It inserts code at the beginning of the function to measure execution counts. Therefore, not much time is taken in compilation.

On the other hand, `gcov` has to do more detailed analysis/instrumentation at compile time as it is trying to measure things like execution counts of each line in the source, stats related to branches, etc. Therefore, `gcov` takes more time to compile.

1.3 Measuring Program Size

Use “`ls -l`” to measure the size of all four versions of binary files generated in the previous section:

Q6- Report the 4 size measurements for the generated binary files. [1 mark]

`-g`	18512 bytes
`-02`	16792 bytes
`-03`	16792 bytes
`-0s`	16792 bytes

Q7-Which size is the smallest and why (describe briefly)? [1 mark]

In the case of the test program test program we used, `-02`, `-03`, `-0s` all had the same (smallest) file sizes. Ideally, you would expect `-0s` to create an executable of the smallest size but given the nature of our test program, `0s` couldn't lower the binary size more than `02` and `03`.

Q8-Which size is the largest and why (describe briefly)? [1 mark]

`-g` is the largest. This is because no optimizations are run. For other optimization levels, some optimisations kick in which help in reducing code (binary) size. Moreover, `-g` flag tells the compiler to generate debugging information that can be used by a debugger to provide more detailed information about the program's execution, such as the values of variables at different points in the program. That added debugging information is the reason of a larger size.

Q9-Based on the results in Q1, which of `gprof` and `gcov` generate smaller binary file and why (describe briefly)? [1 mark]

`gcov`	28304 bytes
`gprof`	17144 bytes

`gprof` generates a smaller binary file.

`gprof` doesn't need to do as much instrumentation of the binary as `gcov`. The reason is the `gprof` collects information regarding time taken for execution and percentage of total time spent in various function. It does so by sampling at runtime. The little instrumentation it does is for measuring execution counts of a function.

On the other hand, `gcov` has to measure execution counts/branch statistics for all lines in the source and hence needs more instrumentation in the binary (counters etc.)

1.4 Measuring Performance

Find the run-time with all four flags used in the previous section:

Q10-Report the 4 execution time measurements. [1 mark]

`-g`	2.542s
`-O2`	2.104s
`-O3`	1.937s
`-Os`	2.509s

Q11-Which version is the slowest and why (describe briefly)? [1 mark]

`-g` is the slowest. No optimisations are run with `-g`. Therefore the execution time is the slowest. Also, the added debugging information makes it slower.

Q12-Which version is the fastest and why (describe briefly)? [1 mark]

`-O3` is the fastest. The maximum number of optimization are run with `-O3` (eg. inlining, licm, loop unrolling etc.). These optimizations are able to reduce the execution time.

Q13-Based on the results in Q1, which of `grof` and `gcov` generates faster binary file and why (describe briefly)? [1 mark]

`gcov`	26.925s
`gprof`	5.362s

`gprof` is faster than `gcov`.

`gprof` does sampling (interrupts program execution) from time to time to see which function is running. This increases execution time as compared to `-g`, `-O2`, `-O3` but not `gcov`.

`gcov` adds a lot of counters etc. to the code as it tries determine execution counts/other stats for each line in the source. This increases the execution time of the program more than `gprof`'s overheads.

1.5 Inspect Assembly

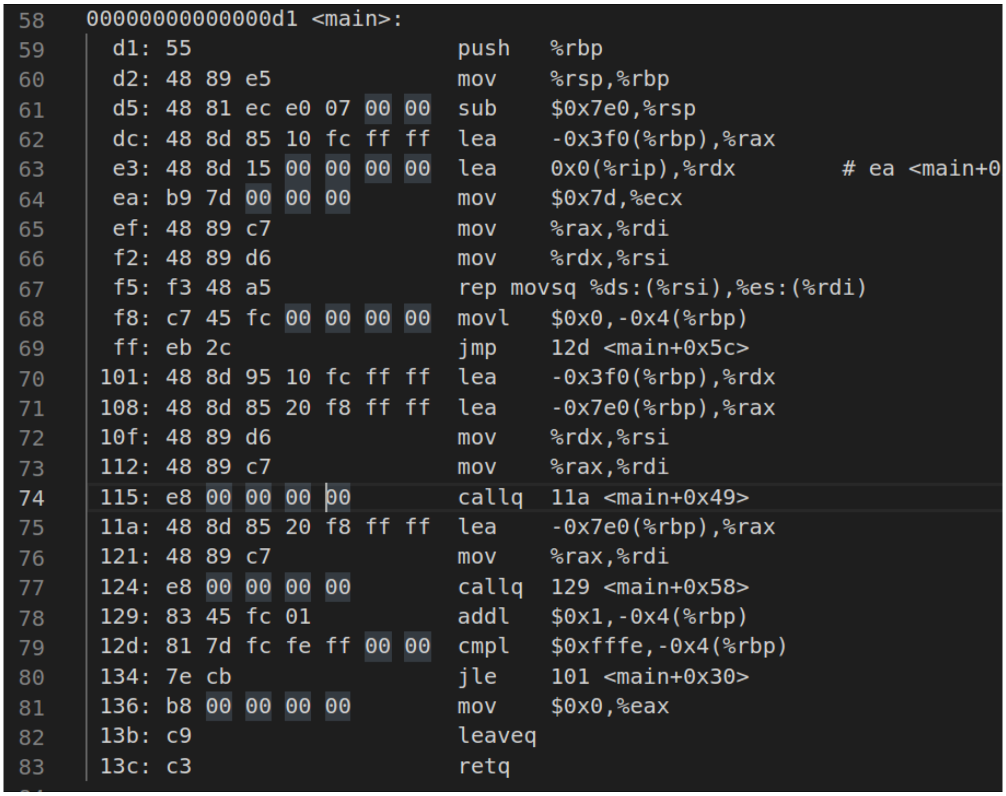
Run the object files created by the compilation flags `-g` and `-O3` to list their assembly instructions for main.c using this command:

```
objdump -d OBJ/main.o
```

Q14- Report the results created by each flag and compare them with each other based on the number of instructions (describe briefly)? [1 mark]

-g:

The `objdump` gives the following assembly. The assembly is unoptimized and can be directly matched to the source code. The main function has only ~34 instructions



```
00000000000000d1 <main>:
  d1: 55                       push   %rbp
  d2: 48 89 e5                 mov    %rsp,%rbp
  d5: 48 81 ec e0 07 00 00     sub    $0x7e0,%rsp
  dc: 48 8d 85 10 fc ff ff     lea    -0x3f0(%rbp),%rax
  e3: 48 8d 15 00 00 00 00     lea    0x0(%rip),%rdx        # ea <main+0
  ea: b9 7d 00 00 00           mov    $0x7d,%ecx
  ef: 48 89 c7                 mov    %rax,%rdi
  f2: 48 89 d6                 mov    %rdx,%rsi
  f5: f3 48 a5                 rep movsq %ds:(%rsi),%es:(%rdi)
  f8: c7 45 fc 00 00 00 00     movl   $0x0,-0x4(%rbp)
  ff: eb 2c                    jmp    12d <main+0x5c>
 101: 48 8d 95 10 fc ff ff     lea    -0x3f0(%rbp),%rdx
 108: 48 8d 85 20 f8 ff ff     lea    -0x7e0(%rbp),%rax
 10f: 48 89 d6                 mov    %rdx,%rsi
 112: 48 89 c7                 mov    %rax,%rdi
 115: e8 00 00 00 00           callq  11a <main+0x49>
 11a: 48 8d 85 20 f8 ff ff     lea    -0x7e0(%rbp),%rax
 121: 48 89 c7                 mov    %rax,%rdi
 124: e8 00 00 00 00           callq  129 <main+0x58>
 129: 83 45 fc 01              addl   $0x1,-0x4(%rbp)
 12d: 81 7d fc fe ff 00 00     cmpl   $0xfffe,-0x4(%rbp)
 134: 7e cb                    jle    101 <main+0x30>
 136: b8 00 00 00 00           mov    $0x0,%eax
 13b: c9                       leaveq
 13c: c3                       retq
```

-O3:

The `objdump` gives the following assembly. The assembly is produced after a lot of optimizations. The number of static instructions is ~390. Example of a notable optimization is loop unrolling.

```
0000000000000000 <main>:
   0: 41 57                 push   %r15
   2: 48 8d 35 00 00 00 00  lea    0x0(%rip),%rsi        # 9 <main+0x9>
   9: b9 7d 00 00 00        mov    $0x7d,%ecx
   e: 41 56                 push   %r14
  10: 41 55                 push   %r13
  12: 41 54                 push   %r12
  14: 55                    push   %rbp
  15: bd ff ff 00 00        mov    $0xffff,%ebp
  1a: 53                    push   %rbx
  1b: bb 46 00 00 00        mov    $0x46,%ebx
  20: 48 81 ec 78 23 00 00  sub    $0x2378,%rsp
  27: 48 89 e7              mov    %rsp,%rdi
  2a: 4c 8d a4 24 f0 03 00  lea    0x3f0(%rsp),%r12
  31: 00
  32: 4c 8d 6c 24 05        lea    0x5(%rsp),%r13
  37: f3 48 a5              rep movsq %ds:(%rsi),%es:(%rdi)
  3a: 66 0f 1f 44 00 00     nopw   0x0(%rax,%rax,1)
  40: 84 db                 test   %bl,%bl
  42: 0f 84 a8 00 00 00     je     f0 <main+0xf0>
  48: e8 00 00 00 00        callq  4d <main+0x4d>
  4d: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
  52: 48 8b 30              mov    (%rax),%rsi
  55: 48 0f be c3           movsbq %bl,%rax
  59: f6 44 46 01 01        testb  $0x1,0x1(%rsi,%rax,2)
  5e: 0f 84 b4 00 00 00     je     118 <main+0x118>
  64: 84 d2                 test   %dl,%dl
  66: 0f 84 34 01 00 00     je     1a0 <main+0x1a0>
  6c: 48 0f be ca           movsbq %dl,%rcx
  70: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
  75: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
  7a: 0f 84 70 01 00 00     je     1f0 <main+0x1f0>
  80: 84 c0                 test   %al,%al
  82: 0f 84 60 02 00 00     je     2e8 <main+0x2e8>
  88: 48 0f be f8           movsbq %al,%rdi
  8c: 48 0f be 4c 24 03     movsbq 0x3(%rsp),%rcx
  92: f6 44 7e 01 01        testb  $0x1,0x1(%rsi,%rdi,2)
  97: 0f 84 2b 02 00 00     je     2c8 <main+0x2c8>
  9d: 84 c9                 test   %cl,%cl
  9f: 0f 84 cf 02 00 00     je     374 <main+0x374>
  a5: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
```

```
   aa: 0f 84 b6 02 00 00     je     366 <main+0x366>
   b0: 4c 8d b4 24 c0 0f 00  lea    0xfc0(%rsp),%r14
   b7: 00
   b8: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
   bd: 4c 89 f7              mov    %r14,%rdi
   c0: e8 00 00 00 00        callq  c5 <main+0xc5>
   c5: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
   ca: e9 b5 02 00 00        jmpq   384 <main+0x384>
   cf: 90                    nop
   d0: 84 d2                 test   %dl,%dl
   d2: 74 1c                 je     f0 <main+0xf0>
   d4: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
   d9: 0f 85 71 03 00 00     jne    450 <main+0x450>
   df: 48 0f be 44 24 04     movsbq 0x4(%rsp),%rax
   e5: 84 c0                 test   %al,%al
   e7: 0f 85 1d 04 00 00     jne    50a <main+0x50a>
   ed: 0f 1f 00              nopl   (%rax)
   f0: c6 84 24 f0 03 00 00  movb   $0x0,0x3f0(%rsp)
   f7: 00
   f8: 4c 89 e7              mov    %r12,%rdi
   fb: e8 00 00 00 00        callq  100 <main+0x100>
  100: 83 ed 01              sub    $0x1,%ebp
  103: 0f 84 cf 00 00 00     je     1d8 <main+0x1d8>
  109: 0f b6 1c 24           movzbl (%rsp),%ebx
  10d: e9 2e ff ff ff        jmpq   40 <main+0x40>
  112: 66 0f 1f 44 00 00     nopw   0x0(%rax,%rax,1)
  118: 84 d2                 test   %dl,%dl
  11a: 74 d4                 je     f0 <main+0xf0>
  11c: 48 0f be ca           movsbq %dl,%rcx
  120: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
  125: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
  12a: 0f 84 10 01 00 00     je     240 <main+0x240>
  130: 84 c0                 test   %al,%al
  132: 0f 84 78 01 00 00     je     2b0 <main+0x2b0>
  138: 48 0f be f8           movsbq %al,%rdi
  13c: 48 0f be 4c 24 03     movsbq 0x3(%rsp),%rcx
  142: f6 44 7e 01 01        testb  $0x1,0x1(%rsi,%rdi,2)
  147: 0f 84 43 01 00 00     je     290 <main+0x290>
  14d: 84 c9                 test   %cl,%cl
  14f: 0f 84 66 02 00 00     je     3bb <main+0x3bb>
  155: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
  15a: 0f 84 4d 02 00 00     je     3ad <main+0x3ad>
  160: 4c 8d bc 24 90 1b 00  lea    0x1b90(%rsp),%r15
  167: 00
  168: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
  16d: 4c 89 ff              mov    %r15,%rdi
  170: e8 00 00 00 00        callq  175 <main+0x175>
  175: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
  17a: e9 4c 02 00 00        jmpq   3cb <main+0x3cb>
  17f: 90                    nop
  180: 84 d2                 test   %dl,%dl
  182: 74 1c                 je     1a0 <main+0x1a0>
```

```
 184: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
 189: 0f 85 65 02 00 00     jne    3f4 <main+0x3f4>
 18f: 48 0f be 44 24 04     movsbq 0x4(%rsp),%rax
 195: 84 c0                 test   %al,%al
 197: 0f 85 de 03 00 00     jne    57b <main+0x57b>
 19d: 0f 1f 00              nopl   (%rax)
 1a0: c6 84 24 e0 07 00 00  movb   $0x0,0x7e0(%rsp)
 1a7: 00
 1a8: 4c 8d b4 24 e0 07 00  lea    0x7e0(%rsp),%r14
 1af: 00
 1b0: 0f be d3              movsbl %bl,%edx
 1b3: 4c 89 e7              mov    %r12,%rdi
 1b6: 31 c0                 xor    %eax,%eax
 1b8: 48 8d 35 00 00 00 00  lea    0x0(%rip),%rsi        # 1bf <main+0x1bf>
 1bf: 4c 89 f1              mov    %r14,%rcx
 1c2: e8 00 00 00 00        callq  1c7 <main+0x1c7>
 1c7: 4c 89 e7              mov    %r12,%rdi
 1ca: e8 00 00 00 00        callq  1cf <main+0x1cf>
 1cf: 83 ed 01              sub    $0x1,%ebp
 1d2: 0f 85 31 ff ff ff     jne    109 <main+0x109>
 1d8: 48 81 c4 78 23 00 00  add    $0x2378,%rsp
 1df: 31 c0                 xor    %eax,%eax
 1e1: 5b                    pop    %rbx
 1e2: 5d                    pop    %rbp
 1e3: 41 5c                 pop    %r12
 1e5: 41 5d                 pop    %r13
 1e7: 41 5e                 pop    %r14
 1e9: 41 5f                 pop    %r15
 1eb: c3                    retq
 1ec: 0f 1f 40 00           nopl   0x0(%rax)
 1f0: 84 c0                 test   %al,%al
 1f2: 74 ac                 je     1a0 <main+0x1a0>
 1f4: 48 0f be c8           movsbq %al,%rcx
 1f8: 48 0f be 54 24 03     movsbq 0x3(%rsp),%rdx
 1fe: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
 203: 0f 84 77 ff ff ff     je     180 <main+0x180>
 209: 84 d2                 test   %dl,%dl
 20b: 0f 84 1a 01 00 00     je     32b <main+0x32b>
 211: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
 216: 0f 84 01 01 00 00     je     31d <main+0x31d>
 21c: 4c 8d bc 24 b0 13 00  lea    0x13b0(%rsp),%r15
 223: 00
 224: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
 229: 4c 89 ff              mov    %r15,%rdi
 22c: e8 00 00 00 00        callq  231 <main+0x231>
 231: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 236: e9 00 01 00 00        jmpq   33b <main+0x33b>
 23b: 0f 1f 44 00 00        nopl   0x0(%rax,%rax,1)
 240: 84 c0                 test   %al,%al
 242: 0f 84 a8 fe ff ff     je     f0 <main+0xf0>
 248: 48 0f be c8           movsbq %al,%rcx
 24c: 48 0f be 54 24 03     movsbq 0x3(%rsp),%rdx
 252: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
 257: 0f 84 73 fe ff ff     je     d0 <main+0xd0>
 25d: 84 d2                 test   %dl,%dl
 25f: 0f 84 e9 00 00 00     je     34e <main+0x34e>
 265: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
 26a: 0f 84 d0 00 00 00     je     340 <main+0x340>
 270: 4c 8d b4 24 80 1f 00  lea    0x1f80(%rsp),%r14
 277: 00
 278: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
 27d: 4c 89 f7              mov    %r14,%rdi
 280: e8 00 00 00 00        callq  285 <main+0x285>
 285: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 28a: e9 cf 00 00 00        jmpq   35e <main+0x35e>
 28f: 90                    nop
 290: 84 c9                 test   %cl,%cl
 292: 74 1c                 je     2b0 <main+0x2b0>
 294: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
 299: 0f 85 73 01 00 00     jne    412 <main+0x412>
 29f: 48 0f be 44 24 04     movsbq 0x4(%rsp),%rax
 2a5: 84 c0                 test   %al,%al
 2a7: 0f 85 7a 02 00 00     jne    527 <main+0x527>
 2ad: 0f 1f 00              nopl   (%rax)
 2b0: c6 84 24 a0 17 00 00  movb   $0x0,0x17a0(%rsp)
 2b7: 00
 2b8: 4c 8d b4 24 a0 17 00  lea    0x17a0(%rsp),%r14
 2bf: 00
 2c0: e9 ee fe ff ff        jmpq   1b3 <main+0x1b3>
 2c5: 0f 1f 00              nopl   (%rax)
 2c8: 84 c9                 test   %cl,%cl
 2ca: 74 1c                 je     2e8 <main+0x2e8>
 2cc: f6 44 4e 01 01        testb  $0x1,0x1(%rsi,%rcx,2)
 2d1: 0f 85 5a 01 00 00     jne    431 <main+0x431>
 2d7: 48 0f be 44 24 04     movsbq 0x4(%rsp),%rax
 2dd: 84 c0                 test   %al,%al
 2df: 0f 85 6c 02 00 00     jne    551 <main+0x551>
 2e5: 0f 1f 00              nopl   (%rax)
 2e8: c6 84 24 d0 0b 00 00  movb   $0x0,0xbd0(%rsp)
 2ef: 00
 2f0: 4c 8d bc 24 d0 0b 00  lea    0xbd0(%rsp),%r15
 2f7: 00
 2f8: 4c 8d b4 24 e0 07 00  lea    0x7e0(%rsp),%r14
 2ff: 00
 300: 4c 89 f9              mov    %r15,%rcx
 303: 48 8d 35 00 00 00 00  lea    0x0(%rip),%rsi        # 30a <main+0x30a>
 30a: 31 c0                 xor    %eax,%eax
 30c: 4c 89 f7              mov    %r14,%rdi
 30f: e8 00 00 00 00        callq  314 <main+0x314>
 314: 0f b6 1c 24           movzbl (%rsp),%ebx
 318: e9 93 fe ff ff        jmpq   1b0 <main+0x1b0>
 31d: 48 0f be 54 24 04     movsbq 0x4(%rsp),%rdx
 323: 84 d2                 test   %dl,%dl
 325: 0f 85 8b 01 00 00     jne    4b6 <main+0x4b6>
```

```
  32b: c6 84 24 b0 13 00 00  movb   $0x0,0x13b0(%rsp)
  332: 00
  333: 4c 8d bc 24 b0 13 00  lea    0x13b0(%rsp),%r15
  33a: 00
  33b: 0f be d0              movsbl %al,%edx
  33e: eb b8                 jmp    2f8 <main+0x2f8>
  340: 48 0f be 54 24 04     movsbq 0x4(%rsp),%rdx
  346: 84 d2                 test   %dl,%dl
  348: 0f 85 92 01 00 00     jne    4e0 <main+0x4e0>
  34e: c6 84 24 80 1f 00 00  movb   $0x0,0x1f80(%rsp)
  355: 00
  356: 4c 8d b4 24 80 1f 00  lea    0x1f80(%rsp),%r14
  35d: 00
  35e: 0f be d0              movsbl %al,%edx
  361: e9 4d fe ff ff        jmpq   1b3 <main+0x1b3>
  366: 48 0f be 54 24 04     movsbq 0x4(%rsp),%rdx
  36c: 84 d2                 test   %dl,%dl
  36e: 0f 85 ee 00 00 00     jne    462 <main+0x462>
  374: c6 84 24 c0 0f 00 00  movb   $0x0,0xfc0(%rsp)
  37b: 00
  37c: 4c 8d b4 24 c0 0f 00  lea    0xfc0(%rsp),%r14
  383: 00
  384: 4c 8d bc 24 d0 0b 00  lea    0xbd0(%rsp),%r15
  38b: 00
  38c: 0f be d0              movsbl %al,%edx
  38f: 31 c0                 xor    %eax,%eax
  391: 4c 89 f1              mov    %r14,%rcx
  394: 48 8d 35 00 00 00 00  lea    0x0(%rip),%rsi        # 39b <main+0x39b>
  39b: 4c 89 ff              mov    %r15,%rdi
  39e: e8 00 00 00 00        callq  3a3 <main+0x3a3>
  3a3: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
  3a8: e9 4b ff ff ff        jmpq   2f8 <main+0x2f8>
  3ad: 48 0f be 54 24 04     movsbq 0x4(%rsp),%rdx
  3b3: 84 d2                 test   %dl,%dl
  3b5: 0f 85 d1 00 00 00     jne    48c <main+0x48c>
  3bb: c6 84 24 90 1b 00 00  movb   $0x0,0x1b90(%rsp)
  3c2: 00
  3c3: 4c 8d bc 24 90 1b 00  lea    0x1b90(%rsp),%r15
  3ca: 00
  3cb: 4c 8d b4 24 a0 17 00  lea    0x17a0(%rsp),%r14
  3d2: 00
  3d3: 0f be d0              movsbl %al,%edx
  3d6: 31 c0                 xor    %eax,%eax
  3d8: 4c 89 f9              mov    %r15,%rcx
  3db: 48 8d 35 00 00 00 00  lea    0x0(%rip),%rsi        # 3e2 <main+0x3e2>
  3e2: 4c 89 f7              mov    %r14,%rdi
  3e5: e8 00 00 00 00        callq  3ea <main+0x3ea>
  3ea: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
  3ef: e9 bf fd ff ff        jmpq   1b3 <main+0x1b3>
  3f4: 4c 8d b4 24 e0 07 00  lea    0x7e0(%rsp),%r14
  3fb: 00
  3fc: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
```

```
 401: 4c 89 f7              mov    %r14,%rdi
 404: e8 00 00 00 00        callq  409 <main+0x409>
 409: 0f b6 1c 24           movzbl (%rsp),%ebx
 40d: e9 9e fd ff ff        jmpq   1b0 <main+0x1b0>
 412: 4c 8d b4 24 a0 17 00  lea    0x17a0(%rsp),%r14
 419: 00
 41a: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
 41f: 4c 89 f7              mov    %r14,%rdi
 422: e8 00 00 00 00        callq  427 <main+0x427>
 427: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
 42c: e9 82 fd ff ff        jmpq   1b3 <main+0x1b3>
 431: 4c 8d bc 24 d0 0b 00  lea    0xbd0(%rsp),%r15
 438: 00
 439: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
 43e: 4c 89 ff              mov    %r15,%rdi
 441: e8 00 00 00 00        callq  446 <main+0x446>
 446: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
 44b: e9 a8 fe ff ff        jmpq   2f8 <main+0x2f8>
 450: 48 8d 74 24 03        lea    0x3(%rsp),%rsi
 455: 4c 89 e7              mov    %r12,%rdi
 458: e8 00 00 00 00        callq  45d <main+0x45d>
 45d: e9 96 fc ff ff        jmpq   f8 <main+0xf8>
 462: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
 467: 0f 84 33 01 00 00     je     5a0 <main+0x5a0>
 46d: 4c 8d b4 24 c0 0f 00  lea    0xfc0(%rsp),%r14
 474: 00
 475: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
 47a: 4c 89 f7              mov    %r14,%rdi
 47d: e8 00 00 00 00        callq  482 <main+0x482>
 482: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 487: e9 f8 fe ff ff        jmpq   384 <main+0x384>
 48c: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
 491: 0f 84 42 01 00 00     je     5d9 <main+0x5d9>
 497: 4c 8d bc 24 90 1b 00  lea    0x1b90(%rsp),%r15
 49e: 00
 49f: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
 4a4: 4c 89 ff              mov    %r15,%rdi
 4a7: e8 00 00 00 00        callq  4ac <main+0x4ac>
 4ac: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 4b1: e9 15 ff ff ff        jmpq   3cb <main+0x3cb>
 4b6: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
 4bb: 0f 84 35 01 00 00     je     5f6 <main+0x5f6>
 4c1: 4c 8d bc 24 b0 13 00  lea    0x13b0(%rsp),%r15
 4c8: 00
 4c9: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
 4ce: 4c 89 ff              mov    %r15,%rdi
 4d1: e8 00 00 00 00        callq  4d6 <main+0x4d6>
 4d6: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 4db: e9 5b fe ff ff        jmpq   33b <main+0x33b>
 4e0: f6 44 56 01 01        testb  $0x1,0x1(%rsi,%rdx,2)
 4e5: 0f 84 28 01 00 00     je     613 <main+0x613>
 4eb: 4c 8d b4 24 80 1f 00  lea    0x1f80(%rsp),%r14
```

```
  4f2: 00
  4f3: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
  4f8: 4c 89 f7              mov    %r14,%rdi
  4fb: e8 00 00 00 00        callq  500 <main+0x500>
  500: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
  505: e9 54 fe ff ff        jmpq   35e <main+0x35e>
  50a: f6 44 46 01 01        testb  $0x1,0x1(%rsi,%rax,2)
  50f: 0f 84 38 01 00 00     je     64d <main+0x64d>
  515: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
  51a: 4c 89 e7              mov    %r12,%rdi
  51d: e8 00 00 00 00        callq  522 <main+0x522>
  522: e9 d1 fb ff ff        jmpq   f8 <main+0xf8>
  527: f6 44 46 01 01        testb  $0x1,0x1(%rsi,%rax,2)
  52c: 0f 84 fe 00 00 00     je     630 <main+0x630>
  532: 4c 8d b4 24 a0 17 00  lea    0x17a0(%rsp),%r14
  539: 00
  53a: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
  53f: 4c 89 f7              mov    %r14,%rdi
  542: e8 00 00 00 00        callq  547 <main+0x547>
  547: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
  54c: e9 62 fc ff ff        jmpq   1b3 <main+0x1b3>
  551: f6 44 46 01 01        testb  $0x1,0x1(%rsi,%rax,2)
  556: 0f 84 01 01 00 00     je     65d <main+0x65d>
  55c: 4c 8d bc 24 d0 0b 00  lea    0xbd0(%rsp),%r15
  563: 00
  564: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
  569: 4c 89 ff              mov    %r15,%rdi
  56c: e8 00 00 00 00        callq  571 <main+0x571>
  571: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
  576: e9 7d fd ff ff        jmpq   2f8 <main+0x2f8>
  57b: f6 44 46 01 01        testb  $0x1,0x1(%rsi,%rax,2)
  580: 74 3b                 je     5bd <main+0x5bd>
  582: 4c 8d b4 24 e0 07 00  lea    0x7e0(%rsp),%r14
  589: 00
  58a: 48 8d 74 24 04        lea    0x4(%rsp),%rsi
  58f: 4c 89 f7              mov    %r14,%rdi
  592: e8 00 00 00 00        callq  597 <main+0x597>
  597: 0f b6 1c 24           movzbl (%rsp),%ebx
  59b: e9 10 fc ff ff        jmpq   1b0 <main+0x1b0>
  5a0: 4c 8d b4 24 c0 0f 00  lea    0xfc0(%rsp),%r14
  5a7: 00
  5a8: 4c 89 ee              mov    %r13,%rsi
  5ab: 4c 89 f7              mov    %r14,%rdi
  5ae: e8 00 00 00 00        callq  5b3 <main+0x5b3>
  5b3: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
  5b8: e9 c7 fd ff ff        jmpq   384 <main+0x384>
  5bd: 4c 8d b4 24 e0 07 00  lea    0x7e0(%rsp),%r14
  5c4: 00
  5c5: 4c 89 ee              mov    %r13,%rsi
  5c8: 4c 89 f7              mov    %r14,%rdi
  5cb: e8 00 00 00 00        callq  5d0 <main+0x5d0>
  5d0: 0f b6 1c 24           movzbl (%rsp),%ebx
```

```
 5d4: e9 d7 fb ff ff        jmpq   1b0 <main+0x1b0>
 5d9: 4c 8d bc 24 90 1b 00  lea    0x1b90(%rsp),%r15
 5e0: 00
 5e1: 4c 89 ee              mov    %r13,%rsi
 5e4: 4c 89 ff              mov    %r15,%rdi
 5e7: e8 00 00 00 00        callq  5ec <main+0x5ec>
 5ec: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 5f1: e9 d5 fd ff ff        jmpq   3cb <main+0x3cb>
 5f6: 4c 8d bc 24 b0 13 00  lea    0x13b0(%rsp),%r15
 5fd: 00
 5fe: 4c 89 ee              mov    %r13,%rsi
 601: 4c 89 ff              mov    %r15,%rdi
 604: e8 00 00 00 00        callq  609 <main+0x609>
 609: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 60e: e9 28 fd ff ff        jmpq   33b <main+0x33b>
 613: 4c 8d b4 24 80 1f 00  lea    0x1f80(%rsp),%r14
 61a: 00
 61b: 4c 89 ee              mov    %r13,%rsi
 61e: 4c 89 f7              mov    %r14,%rdi
 621: e8 00 00 00 00        callq  626 <main+0x626>
 626: 0f b6 44 24 02        movzbl 0x2(%rsp),%eax
 62b: e9 2e fd ff ff        jmpq   35e <main+0x35e>
 630: 4c 8d b4 24 a0 17 00  lea    0x17a0(%rsp),%r14
 637: 00
 638: 4c 89 ee              mov    %r13,%rsi
 63b: 4c 89 f7              mov    %r14,%rdi
 63e: e8 00 00 00 00        callq  643 <main+0x643>
 643: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
 648: e9 66 fb ff ff        jmpq   1b3 <main+0x1b3>
 64d: 4c 89 ee              mov    %r13,%rsi
 650: 4c 89 e7              mov    %r12,%rdi
 653: e8 00 00 00 00        callq  658 <main+0x658>
 658: e9 9b fa ff ff        jmpq   f8 <main+0xf8>
 65d: 4c 8d bc 24 d0 0b 00  lea    0xbd0(%rsp),%r15
 664: 00
 665: 4c 89 ee              mov    %r13,%rsi
 668: 4c 89 ff              mov    %r15,%rdi
 66b: e8 00 00 00 00        callq  670 <main+0x670>
 670: 0f be 54 24 01        movsbl 0x1(%rsp),%edx
 675: e9 7e fc ff ff        jmpq   2f8 <main+0x2f8>
```

(Optional Question)

Q15- Name the shortest GCC compiler flag (i.e, -xx) to enable a compiler optimization that requires memory alignment. How many bytes does the data need to be aligned? [1 bonus Mark]

`-O2`.

The alignment of data depends on the size of the variable and the computer architecture.

Most computer requires data alignment based on the size of the data, i.e. an variable of size K must starts at an address that is a multiple of K.

There could be other alignment requirement based on different computer architecture.