HOMEWORK 3#

# 1st Part:

1st Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.618794
-O1	16.784 bytes	46.368701
-O2	16.784 bytes	44.427056
-O3	16.784 bytes	43.905783
-Ofast	18.440 bytes	44.456568

2nd Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	104.122806
-O1	16.784 bytes	47.120258
-O2	16.784 bytes	43.878497
-O3	16.784 bytes	43.939515
-Ofast	18.440 bytes	43.719191

3rd Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.280955
-O1	16.784 bytes	46.505460
-O2	16.784 bytes	43.292906
-O3	16.784 bytes	43.936632
-Ofast	18.440 bytes	43.955931

4th Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.461819
-O1	16.784 bytes	46.445464
-O2	16.784 bytes	43.936902
-O3	16.784 bytes	44.214681
-Ofast	18.440 bytes	44.245843

5th Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.612881
-O1	16.784 bytes	46.455280
-O2	16.784 bytes	44.701636
-O3	16.784 bytes	44.106878
-Ofast	18.440 bytes	44.317558

6th Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.760877
-O1	16.784 bytes	46.252709
-O2	16.784 bytes	43.866798
-O3	16.784 bytes	44.256703
-Ofast	18.440 bytes	43.817896

7th Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.360368
-O1	16.784 bytes	47.914719
-O2	16.784 bytes	44.211739
-O3	16.784 bytes	44.280086
-Ofast	18.440 bytes	44.284794

8th Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.473882
-O1	16.784 bytes	46.414542
-O2	16.784 bytes	44.323836
-O3	16.784 bytes	44.432645
-Ofast	18.440 bytes	43.989896

9th Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	105.411050
-O1	16.784 bytes	46.686785
-O2	16.784 bytes	43.715876
-O3	16.784 bytes	43.832360
-Ofast	18.440 bytes	44.117105

10th Experiment:

Optimization Level	Executable Size	Execution Time
-O0	16.784 bytes	103.937413
-O1	16.784 bytes	46.689729
-O2	16.784 bytes	44.312777
-O3	16.784 bytes	44.370095
-Ofast	18.440 bytes	43.936183

Average Values:

Optimization Level	Average Executable Size	Average Execution Time
-O0	16.784 bytes	103.8040845
-O1	16.784 bytes	46.6853647
-O2	16.784 bytes	44.0668023
-O3	16.784 bytes	43.0885473
-Ofast	18.440 bytes	44.0840965

Technical Information: In these experiments, pc with intel core i7 7500U, 4 mb cache and 12 gb memory is used. In addition, version of GCC is 8.3.0

Explanation: In order to understand results of these experiments, we have to know that “What are the GCC optimization levels?” and “What are the working principles of them?”.

- -O0: At this optimization level, GCC does not perform any optimization and compiles the source code in the most straightforward way possible. Each command in the source code is converted directly to the corresponding instructions in the executable file without rearrangement. If we look at the tables, -O0 has the highest execution times for each experiment. There is a huge difference between -O0 and the nearest optimization level.

- O1: This level turns on the most common forms of optimization that do not require any speed-space tradeoffs.With this option, the resulting executables should be faster than -O0. The more expensive optimizations such as instruction scheduling, are not used at this level. In our experiment, the time difference between -O0 and -O1 is approximately 50% of -O0.
- O2: This option turns on further optimizations, in addition to those used by O1. These additional optimizations include instruction scheduling. Only optimizations that do not require any speed-space tradeoffs are used, so the executable should not increase in size. Our executable size is not changed. On the other hand, further optimizations reduced execution time.
- -O3: This optimization turns on more expensive optimizations, such as function inlining, loop unrolling, in addition to all the optimizations of the lower levels -O2 and -O1. As we can see, execution time is reduced with this optimization.
- -Ofast: -Ofast disregards strict standart compliance. In this optimization, execution time is increased, if we compare with -O3. In addition, executable size is increased. If we working with embedded systems, we don’t want to increase the size because of limited resource.

# 2nd Part:

1st Experiment:

Optimized Version	Executable Size	Execution Time
unoptimized	16784 bytes	103.618794
unoptimized_o3	16784 bytes	43.905783
no_fc (Function call eliminated)	16760 bytes	91.055731
no_fc_o3	16760 bytes	45.001418
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.295109
no_fc_temp_o3	16760 bytes	47.352869
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.092897
no_fc_temp_unrolled_o3	16776 bytes	47.448695

2nd Experiment:

Optimized Version	Executable Size	Execution Time
unoptimized	16784 bytes	104.122806
unoptimized_o3	16784 bytes	43.939515
no_fc (Function call eliminated)	16760 bytes	89.864559
no_fc_o3	16760 bytes	44.992713
no_fc_temp (Function call eliminated + accumulated in	16760 bytes	76.900840

temporary)		
no_fc_temp_o3	16760 bytes	47.448997
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	69.895808
no_fc_temp_unrolled_o3	16776 bytes	47.244103

3rd Experiment:

Optimized Version	**Executable Size**	**Execution Time**
unoptimized	16784 bytes	103.280955
unoptimized_o3	16784 bytes	43.936632
no_fc (Function call eliminated)	16760 bytes	90.483653
no_fc_o3	16760 bytes	43.588521
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.454923
no_fc_temp_o3	16760 bytes	47.764209
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.760515
no_fc_temp_unrolled_o3	16776 bytes	47.112899

4th Experiment:

Optimized Version	**Executable Size**	**Execution Time**
unoptimized	16784 bytes	103.461819
unoptimized_o3	16784 bytes	44.214681
no_fc (Function call eliminated)	16760 bytes	90.816540
no_fc_o3	16760 bytes	43.714263
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.253844
no_fc_temp_o3	16760 bytes	47.295646
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.848205
no_fc_temp_unrolled_o3	16776 bytes	47.585100

5th Experiment:

Optimized Version	Executable Size	Execution Time
unoptimized	16784 bytes	103.612881
unoptimized_o3	16784 bytes	44.106878
no_fc (Function call eliminated)	16760 bytes	90.754564
no_fc_o3	16760 bytes	44.328880
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.274213
no_fc_temp_o3	16760 bytes	47.345341
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.347190
no_fc_temp_unrolled_o3	16776 bytes	47.383301

6th Experiment:

Optimized Version	Executable Size	Execution Time
unoptimized	16784 bytes	103.760877
unoptimized_o3	16784 bytes	44.256703
no_fc (Function call eliminated)	16760 bytes	90.676369
no_fc_o3	16760 bytes	44.386848
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.218425
no_fc_temp_o3	16760 bytes	47.311651
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.759149
no_fc_temp_unrolled_o3	16776 bytes	47.875905

7th Experiment:

Optimized Version	Executable Size	Execution Time
unoptimized	16784 bytes	103.360368
unoptimized_o3	16784 bytes	44.280086
no_fc (Function call eliminated)	16760 bytes	90.633335

no_fc_o3	16760 bytes	44.346939
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.458558
no_fc_temp_o3	16760 bytes	47.488330
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.825291
no_fc_temp_unrolled_o3	16776 bytes	47.447711

8th Experiment:

Optimized Version	**Executable Size**	**Execution Time**
unoptimized	16784 bytes	103.473882
unoptimized_o3	16784 bytes	44.432645
no_fc (Function call eliminated)	16760 bytes	90.757619
no_fc_o3	16760 bytes	44.187469
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.044549
no_fc_temp_o3	16760 bytes	47.470690
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	71.114747
no_fc_temp_unrolled_o3	16776 bytes	47.185602

9th Experiment:

Optimized Version	**Executable Size**	**Execution Time**
unoptimized	16784 bytes	105.411050
unoptimized_o3	16784 bytes	43.832360
no_fc (Function call eliminated)	16760 bytes	90.505432
no_fc_o3	16760 bytes	44.374764
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	76.909115
no_fc_temp_o3	16760 bytes	47.453629
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.536538

no_fc_temp_unrolled_o3	16776 bytes	47.191364

10th Experiment:

Optimized Version	Executable Size	Execution Time
unoptimized	16784 bytes	103.937413
unoptimized_o3	16784 bytes	44.370095
no_fc (Function call eliminated)	16760 bytes	90.462114
no_fc_o3	16760 bytes	44.309963
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	76.857320
no_fc_temp_o3	16760 bytes	47.469056
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.848795
no_fc_temp_unrolled_o3	16776 bytes	47.183604

Average Values:

Optimized Version	Average Executable Size	Average Execution Time
unoptimized	16784 bytes	103.8040845
unoptimized_o3	16784 bytes	43.0885473
no_fc (Function call eliminated)	16760 bytes	90.6009916
no_fc_o3	16760 bytes	44.3231778
no_fc_temp (Function call eliminated + accumulated in temporary)	16760 bytes	77.1666896
no_fc_temp_o3	16760 bytes	47.4400418
no_fc_temp_unrolled (Function call eliminated + accumulated in temporary + loop unrolled)	16776 bytes	70.6029135
no_fc_temp_unrolled_o3	16776 bytes	47.3658284

Technical Information: In these experiments, pc with intel core i7 7500U, 4 mb cache and 12 gb memory is used. In addition, version of GCC is 8.3.0

Explanation:

- In unoptimized version, GCC does not perform any optimization and compiles the source code in the most straightforward way possible. Each command in the source code is converted directly to the corresponding instructions in the executable file without rearrangement. So, we can see that the highest execution time values belong to unoptimized version in our tables.

- We know that -O3 reduced the execution time from the 1st part.
- 1st Part is related to GCC compiler optimization. Using GCC optimization levels, we can reduce the execution time automatically. However the questions are : "What can we do for optimizing the code manually?" "Can we modify the code in better way in order to reduce execution time?". That's why under this sentences, we'll see some techniques that decrease execution time.
- Function call eliminated: In this technique, we removed the function which is called "muld" . We placed the code inside this function into the line it is called. We observe that execution time is decreased approximately 10 seconds with elimination, if we compare the unoptimized version. So function call is costly operation. We know that in assembly for every calling there is a jump over there. On the other hand , executable size is decreased because of removing the piece of code. There are limited resources in embedded systems. With -O3 , there is nearly no difference between unoptimized -O3 and this.Probably in this case, -O3 can did the best. There is nothing more to do.
- + Accumulated in temporary: In this technique, instead of storing intermidiate results in res[i][j], we defined a temporary variable and accumulated the values in this variable, then updated the array element at the end of the loop. Here we observe that the execution time is reduced again if we compare the previous operation. Because working with registers is faster than working with memory. All operation is done in the register and at the end, result is sent to the memory.
- + Loop unrolled: The goal of loop unrolling is to increase the program's speed by reducing or eliminating instructions that control the loop such as "end of loop" test for each iteration. The best values without -O3 are observed here. However, program code size is increased, which can be undesirable particularly for embedded.
- As we can see, -O3 values are nearly the same. Because -O3 did the best. There is nothing more to do.In addition all of these, we did three different operation and we didn't catch the -O3 values. GCC optimization does its job perfectly.