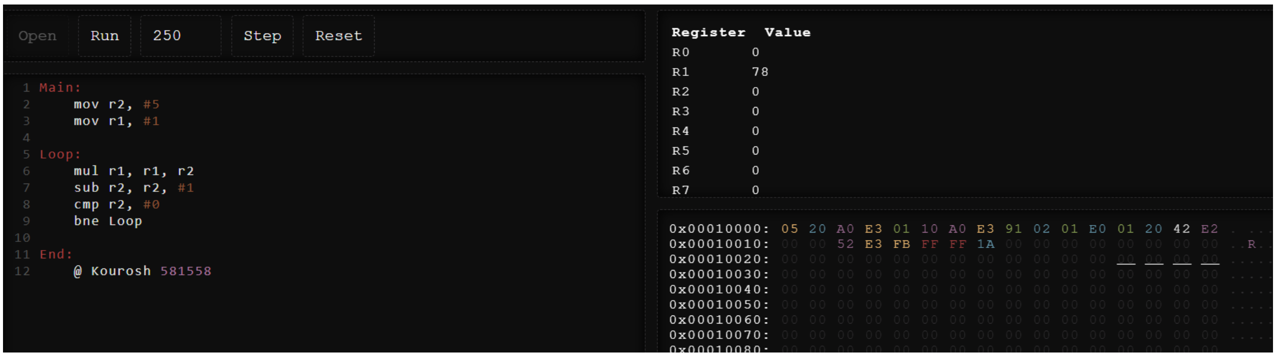
Template Week 4 – Software

Student number: 581558

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



Assignment 4.2: Programming languages

Take screenshots that the following commands work:

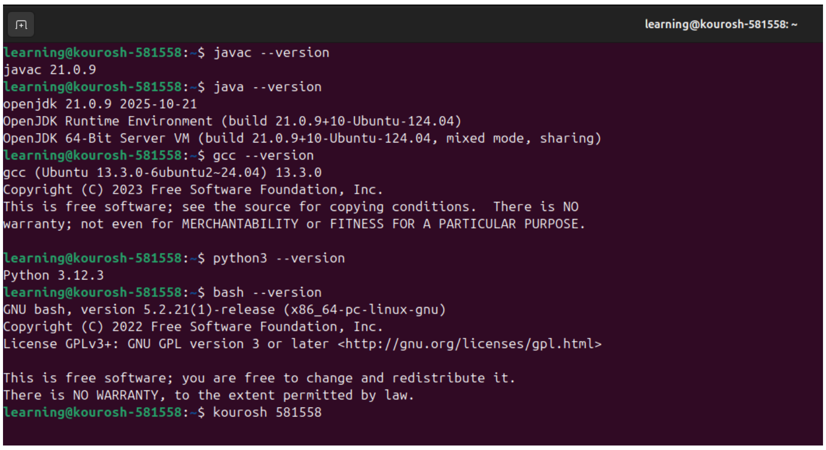
javac --version

java --version

gcc --version

python3 –version

bash --version



Assignment 4.3: Compile

Which of the above files need to be compiled before you can run them?

fib.c and Fibonacci.java

Which source code files are compiled into machine code and then directly executable by a processor?

fib.c

Which source code files are compiled to byte code?

Fibonacci.java

Which source code files are interpreted by an interpreter?

fib.py

These source code files will perform the same calculation after compilation/interpretation. Which one is expected to do the calculation the fastest?

fib.c

How do I run a Java program?

- *javac Fibonacci.java*
- *java Fibonacci*

How do I run a Python program?

- *python3 fib.py*

How do I run a C program?

- *gcc fib.c -o fibProgram*
- *fibProgram.exe (in windows) or ./fibProgram (in linux)*

How do I run a Bash script?

- *chmod a+x fib.sh*
- *bash fib.sh*

If I compile the above source code, will a new file be created? If so, which file?

Gcc → Yes, it creates an executable file

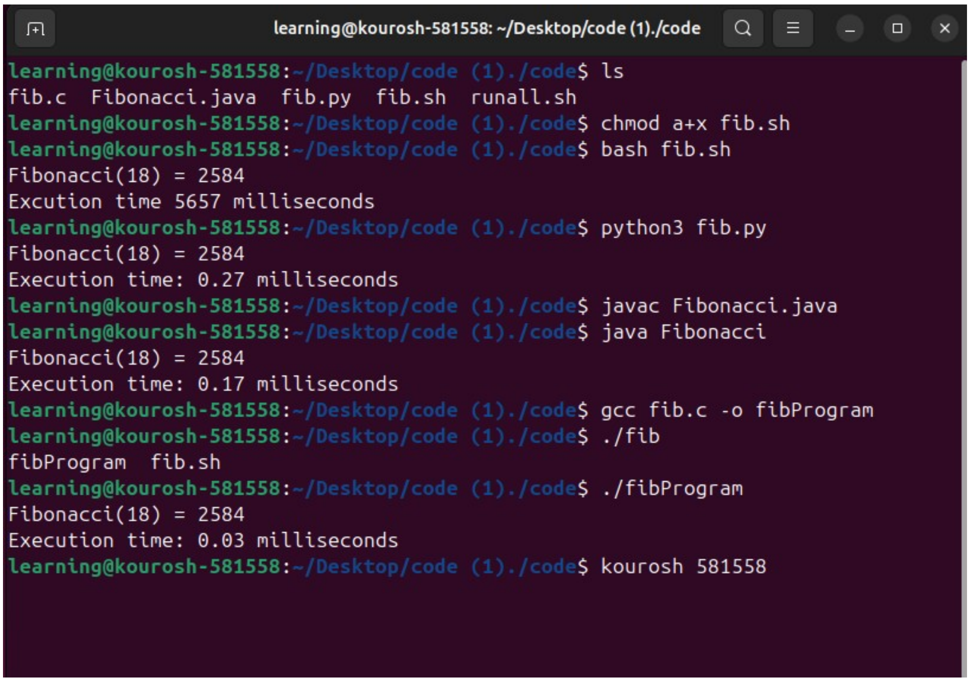
Javac → Yes, it creates a bytecode file

Python3 → No

Bash → No

Take relevant screenshots of the following commands:

- Compile the source files where necessary
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?



GCC compiled it the fastest with only 0.03 milliseconds.

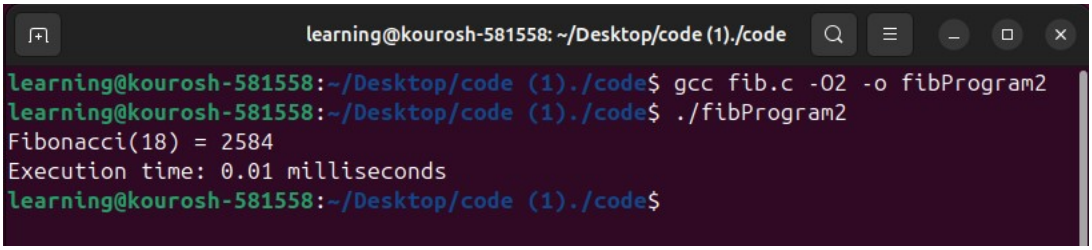
Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

a) Figure out which parameters you need to pass to **the gcc** compiler so that the compiler performs a number of optimizations that will ensure that the compiled source code will run faster. **Tip!** The parameters are usually a letter followed by a number. Also read **page 191** of your book, but find a better optimization in the man pages. Please note that Linux is case sensitive.

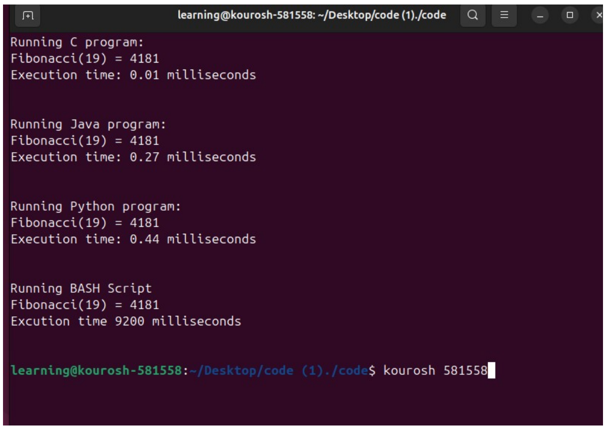
b) Compile **fib.c** again with the optimization parameters

c) Run the newly compiled program. Is it true that it now performs the calculation faster?
Yes, O2 optimization reduced its execution time by 0.02 milliseconds from 0.03 ms to 0.01 ms



```
learning@kourosh-581558: ~/Desktop/code (1)./code
learning@kourosh-581558:~/Desktop/code (1)./code$ gcc fib.c -O2 -o fibProgram2
learning@kourosh-581558:~/Desktop/code (1)./code$ ./fibProgram2
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
learning@kourosh-581558:~/Desktop/code (1)./code$
```

d) Edit the file **runall.sh**, so you can perform all four calculations in a row using this Bash script. So the (compiled/interpreted) C, Java, Python and Bash versions of Fibonacci one after the other.



```
learning@kourosh-581558: ~/Desktop/code (1)./code
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.27 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.44 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 9200 milliseconds


learning@kourosh-581558:~/Desktop/code (1)./code$ kourosh 581558
```

Assignment 4.5: More ARM Assembly

Like the factorial example, you can also implement the calculation of a power of 2 in assembly. For example you want to calculate $2^4 = 16$. Use iteration to calculate the result. Store the result in r0.

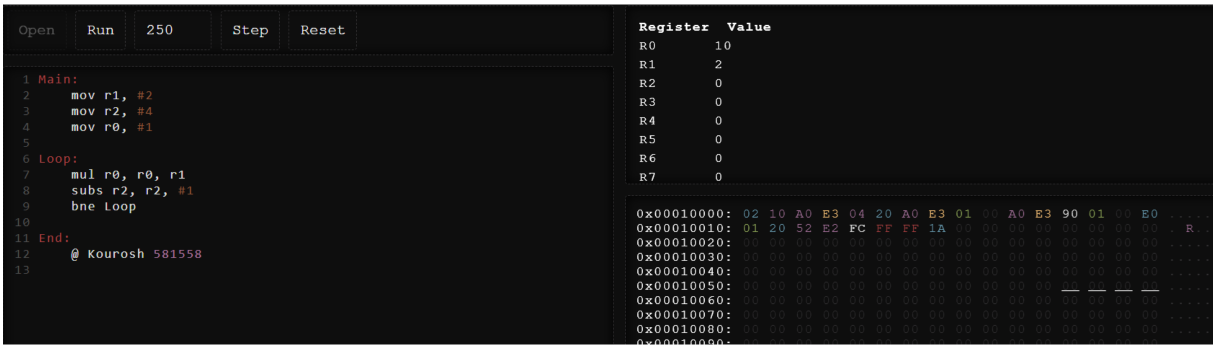
```
Main:
mov r1, #2
mov r2, #4


Loop:


End:
```

Complete the code. See the PowerPoint slides of week 4.

Screenshot of the completed code here.



Ready? Save this file and export it as a pdf file with the name: **week4.pdf**