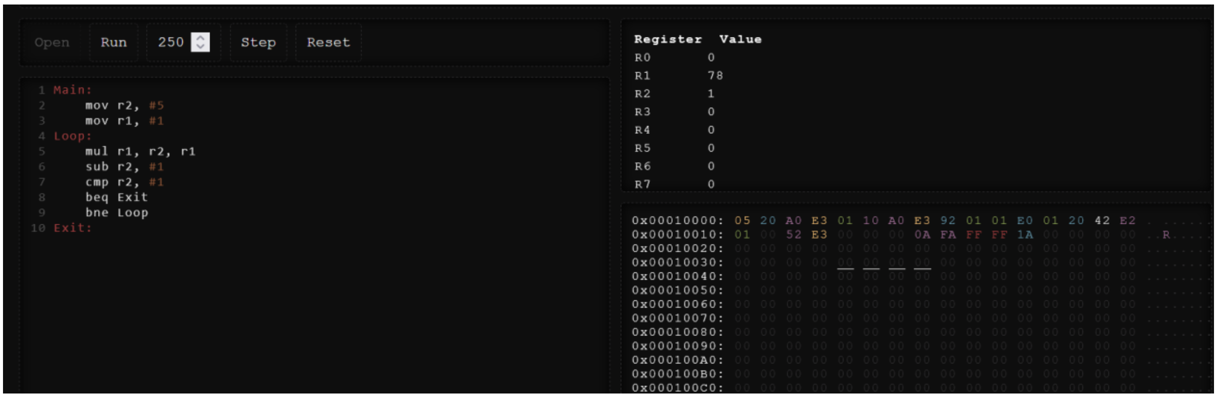
Week 4 – Software

Student number: 579444

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
    mov r2, #5
    mov r1, #1
Loop:
    mul r1, r2, r1
    sub r2, #1
    cmp r2, #1
    beq Exit
    bne Loop
Exit:
```

```
Register  Value
R0        0
R1        78
R2        1
R3        0
R4        0
R5        0
R6        0
R7        0
```

```
0x00010000: 05 20 A0 E3 01 10 A0 E3 92 01 01 E0 01 20 42 E2 ........
0x00010010: 01 00 52 E3 00 00 00 0A FA FF FF 1A 00 00 00 00 ..R.....
0x00010020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x00010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x00010040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x00010050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x00010060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x00010070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x00010080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x00010090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x000100A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x000100B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
0x000100C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

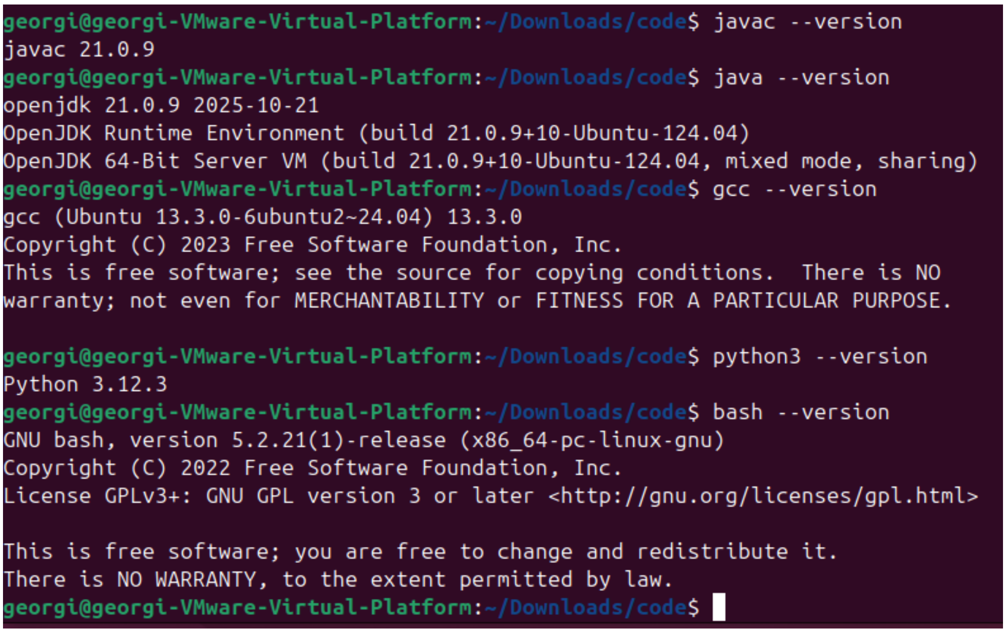
javac --version

java --version

gcc --version

python3 --version

bash --version



```
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ javac --version
javac 21.0.9
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ java --version
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ python3 --version
Python 3.12.3
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ bash --version
GNU bash, version 5.2.21(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ 
```

Assignment 4.3: Compile

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

The java and c files must be compiled before executing them.

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

The python and bash files are compiled into machine code and then directly executable.

Which source code files are compiled to byte code?

Java and C are compiled to byte code.

Which source code files are interpreted by an interpreter?

Python and Bash are interpreted by an interpreter.

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

The c file will do the calculation the fastest.

How do I run a Java program?

Using the javac command to first compile it, then running the java “nameOfFile” command.

How do I run a Python program?

Using the python3 command – it doesn’t need to be compiled beforehand.

How do I run a C program?

Using the gcc command first to compile the code, then executing the executable file made from the compiling.

How do I run a Bash script?

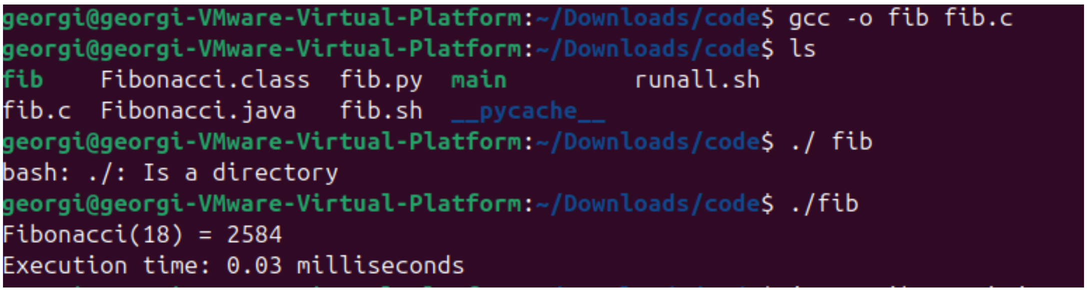
Using the bash command or just ./”nameOfFile”.

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

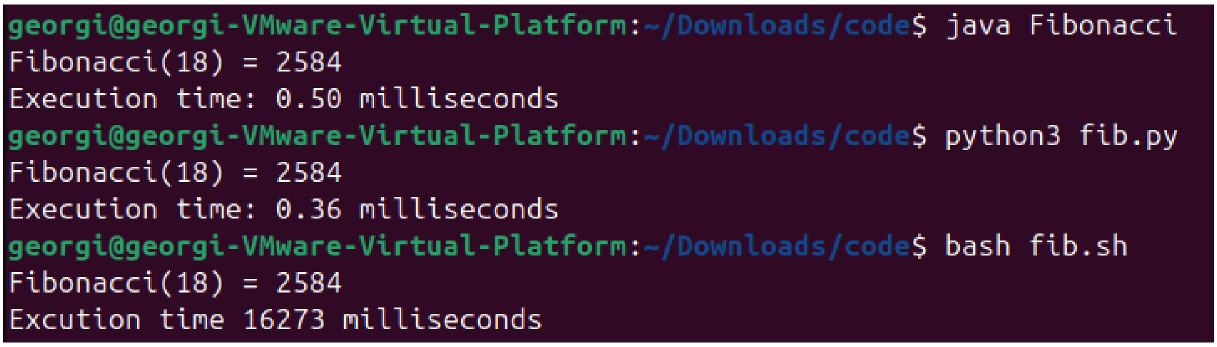
Yes, a .class file will be created for the java file and an executable file will be created for the c file.

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?



```
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ gcc -o fib fib.c
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib     Fibonacci.class  fib.py  main         runall.sh
fib.c   Fibonacci.java   fib.sh  __pycache__
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ ./ fib
bash: ./: Is a directory
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
```

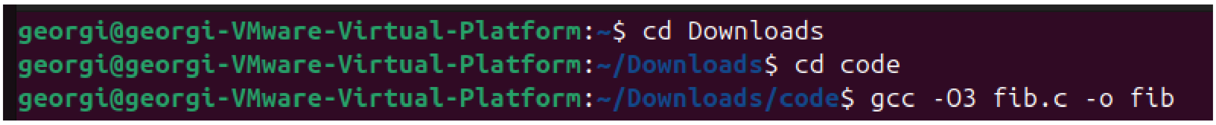


```
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.50 milliseconds
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.36 milliseconds
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ bash fib.sh
Fibonacci(18) = 2584
Excution time 16273 milliseconds
```

Assignment 4.4: Optimize

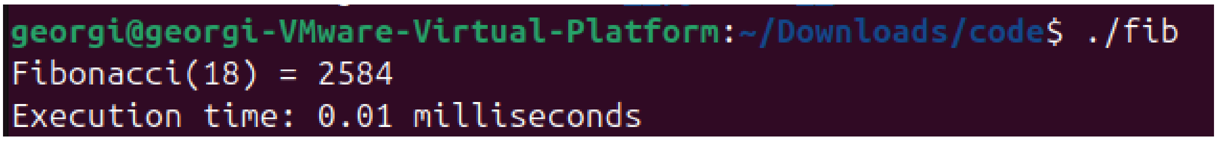
Take relevant screenshots of the following commands:

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.



```
georgi@georgi-VMware-Virtual-Platform:~$ cd Downloads
georgi@georgi-VMware-Virtual-Platform:~/Downloads$ cd code
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 fib.c -o fib
```

b) Compile fib.c again with the optimization parameters

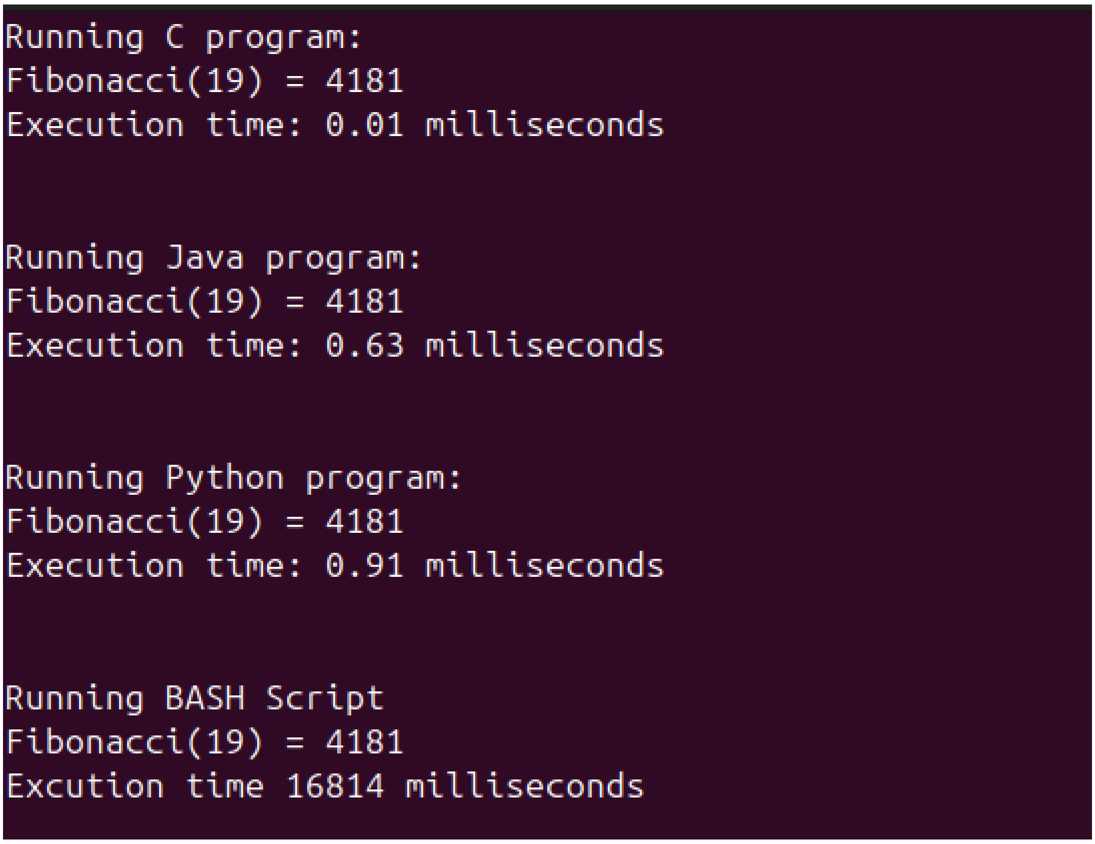


```
georgi@georgi-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
```

c) Run the newly compiled program. Is it true that it now performs the calculation faster?

Yes, it went down from 0.03 milliseconds to 0.01 milliseconds.

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.



```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.63 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.91 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 16814 milliseconds
```

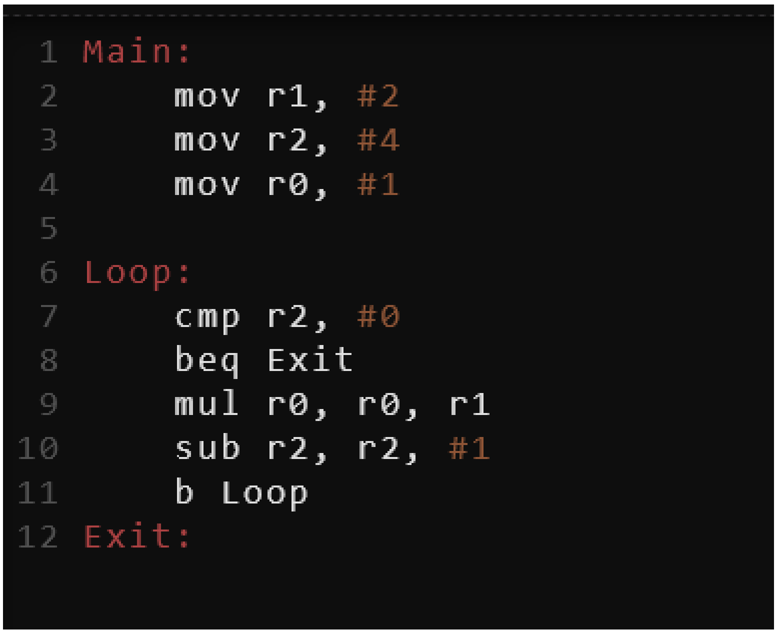
Assignment 4.5: More ARM Assembly

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.



```
Main:
    mov r1, #2
    mov r2, #4
    mov r0, #1

Loop:
    cmp r2, #0
    beq Exit
    mul r0, r0, r1
    sub r2, r2, #1
    b Loop
Exit:
```

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