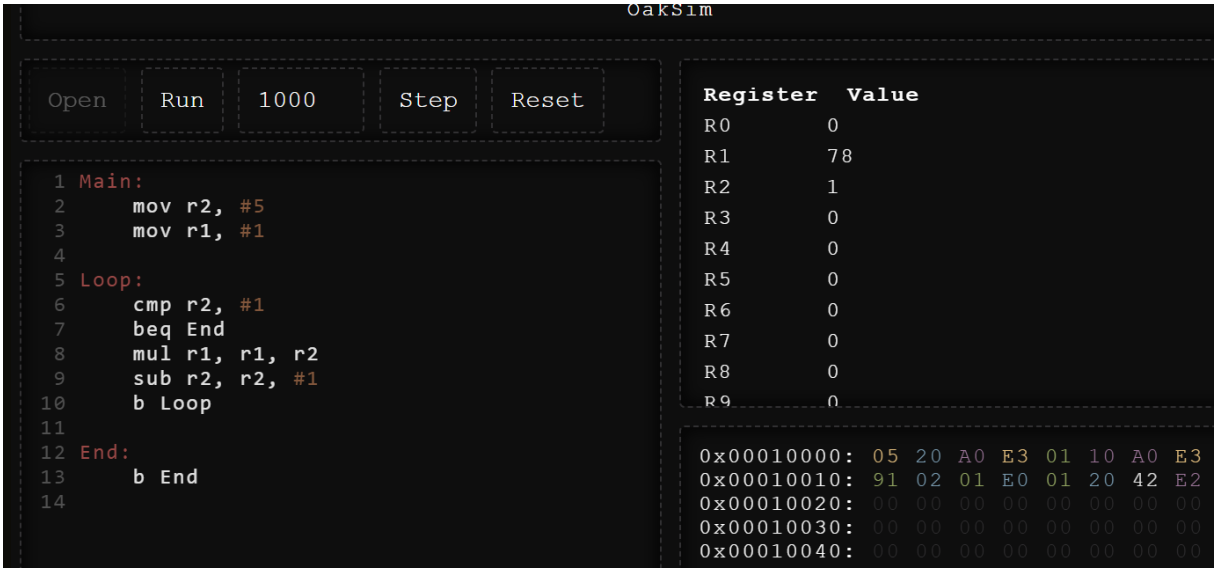
Template Week 4 – Software

Student number:568261

Assignment 4.1: ARM assembly

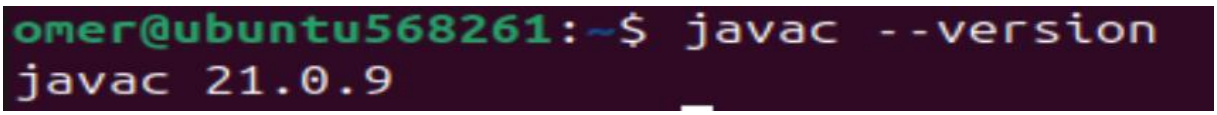
Screenshot of working assembly code of factorial calculation:



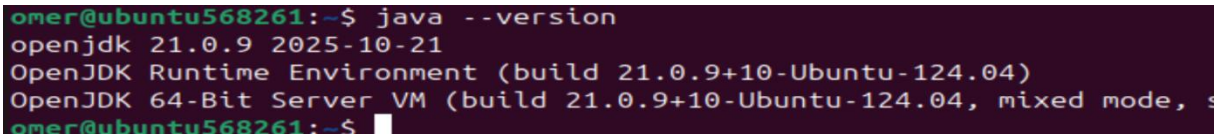
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

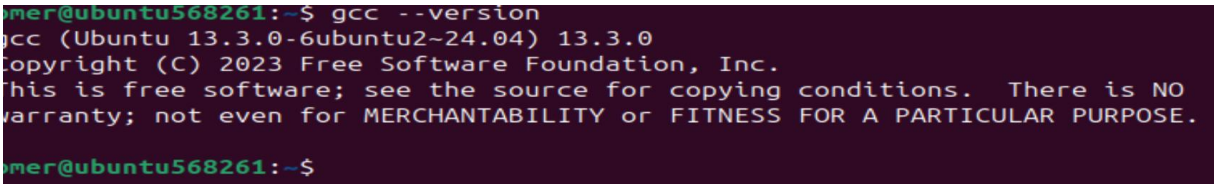
javac –version



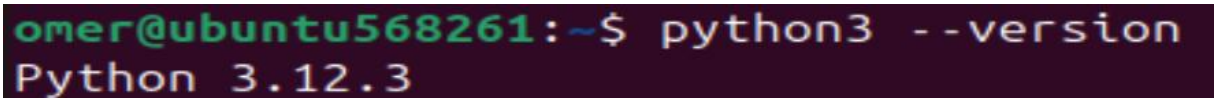
java –version



gcc –version



python3 –version



bash --version

Assignment 4.3: Compile

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

Fibonacci.java

Fib.c

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

fib.sh

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 fib

./fib

How do I run a Bash script?

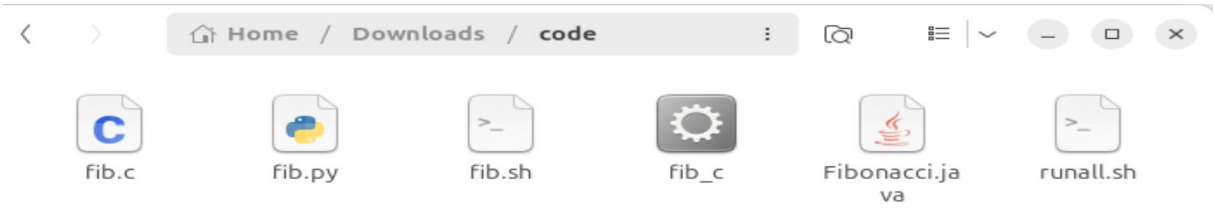
bash fib.sh

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

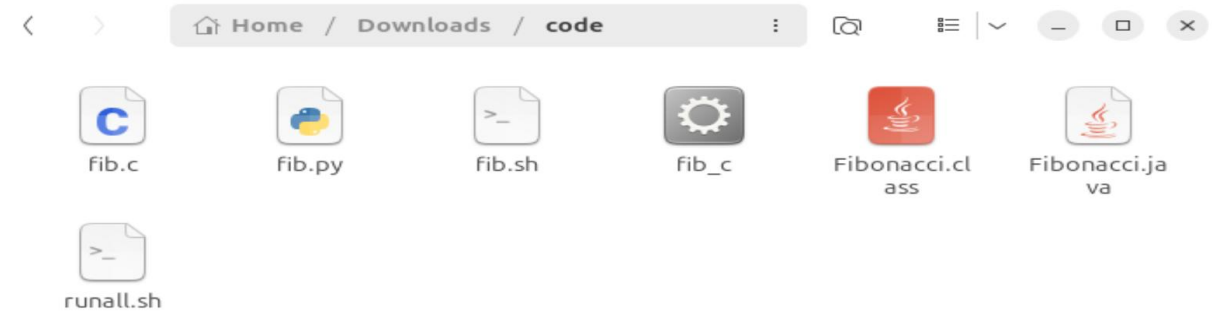
- Fibonacci.java → Fibonacci.class
- fib.c → fib (of fib.exe)

Take relevant screenshots of the following commands:

- Compile the source files where necessary



- Make them executable



- Run them

```
omer@ubuntu568261:~/Downloads/code$ ./fib_c
Fibonacci(18) = 2584
Execution time: 0.05 milliseconds
omer@ubuntu568261:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.85 milliseconds
omer@ubuntu568261:~/Downloads/code$ python3 fib.py
\Fibonacci(18) = 2584
Execution time: 1.14 milliseconds
omer@ubuntu568261:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 15763 milliseconds
omer@ubuntu568261:~/Downloads/code$
```

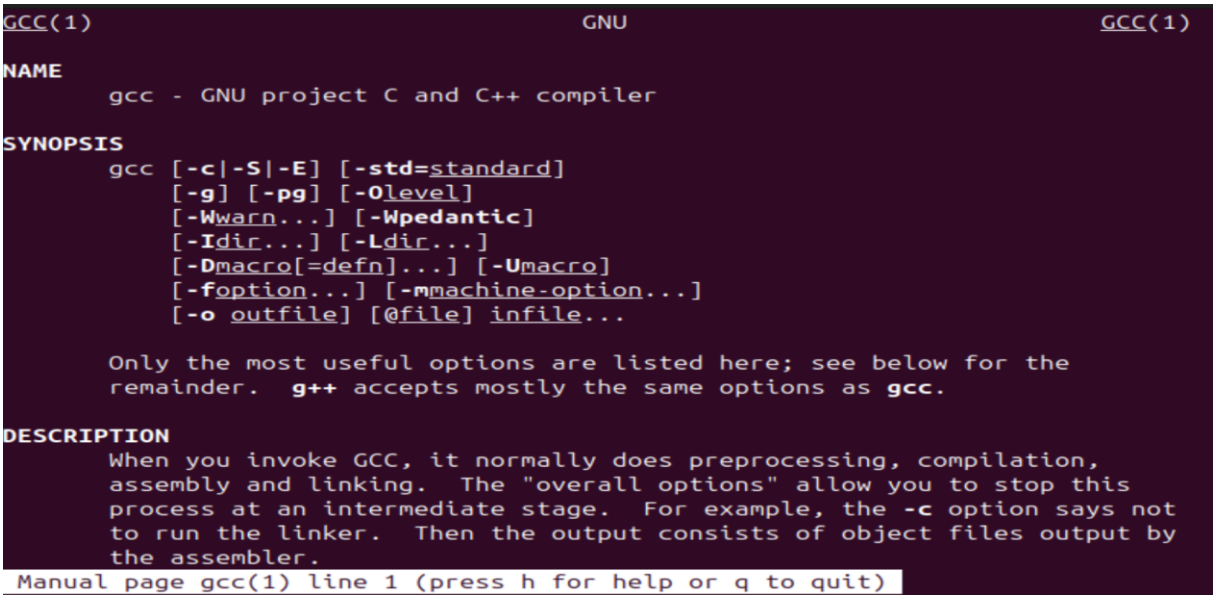
- Which (compiled) source code file performs the calculation the fastest?

Fib.C

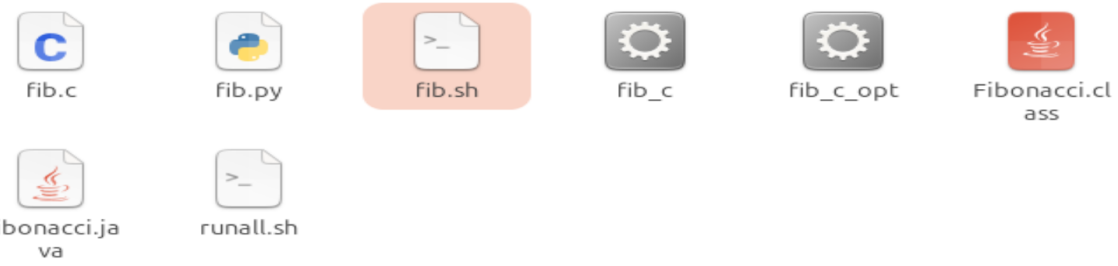
Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

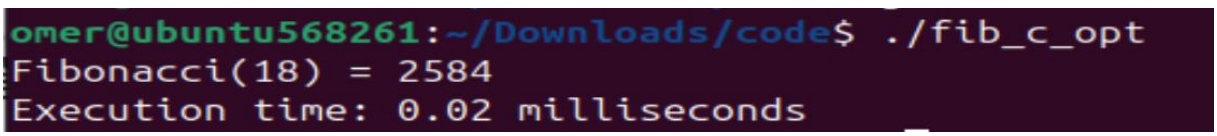
a) Figure out which parameters you need to pass to **the gcc** compiler so that the compiler performs a number of optimizations that will ensure that the compiled source code will run faster. **Tip!** The parameters are usually a letter followed by a number. Also read **page 191** of your book, but find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



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

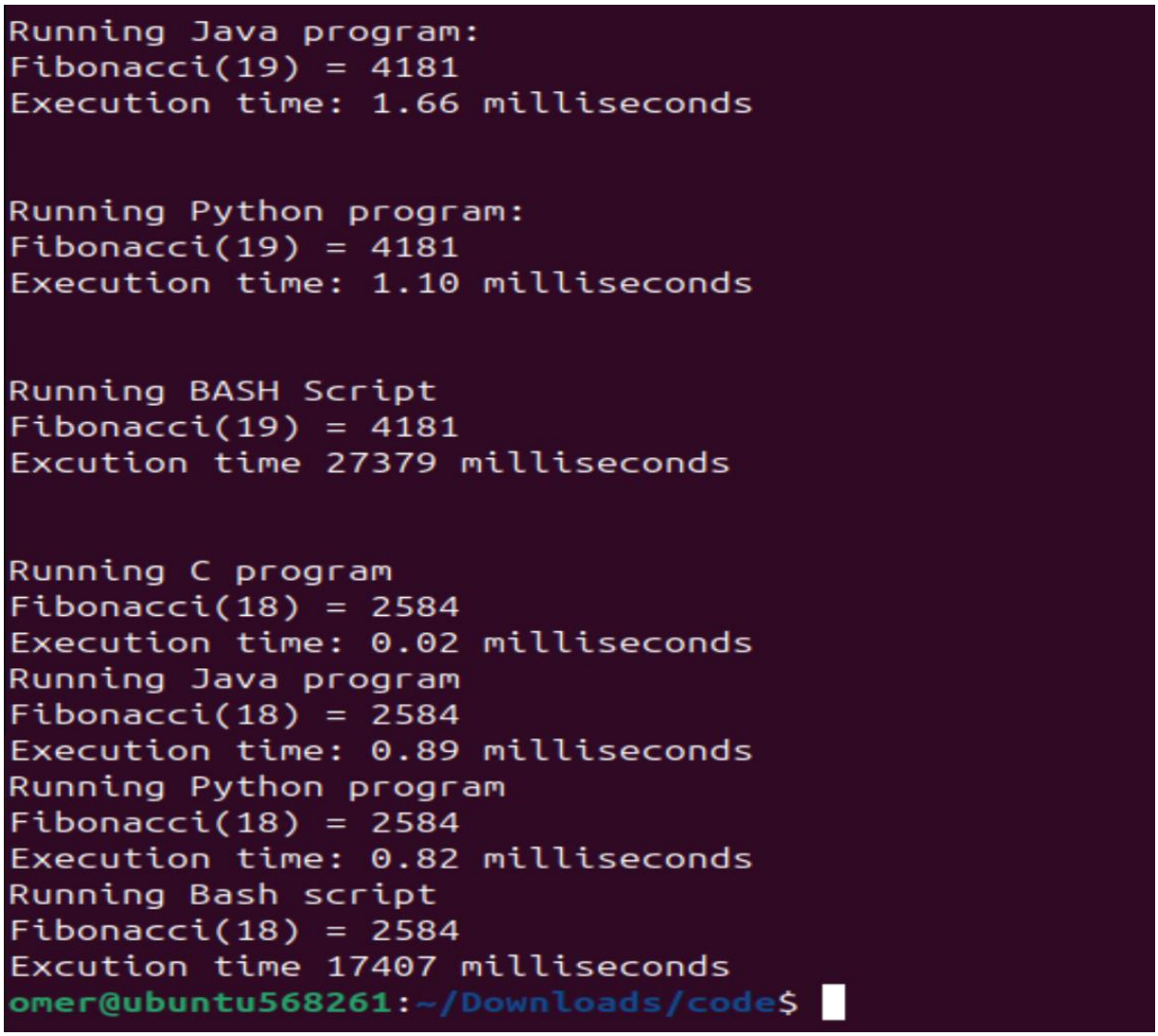


c) Run the newly compiled program. Is it true that it now performs the calculation faster?



Yes it runs faster

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.



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      @ base = 2
    mov r2, #4      @ exponent = 4
    mov r0, #1      @ result = 1
```

Loop:

```
    mul r0, r0, r1  @ result = result * 2
    subs r2, r2, #1 @ exponent--
    bne Loop        @ repeat if exponent != 0
```

End:

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