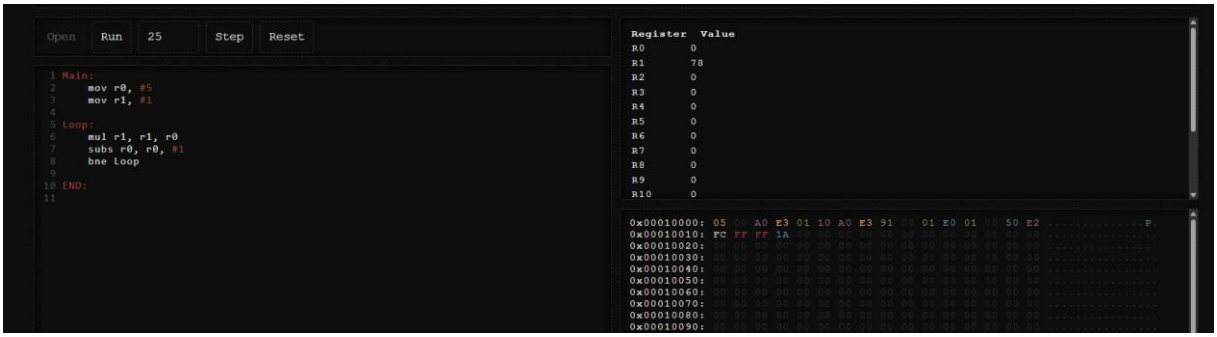
Template Week 4 – Software

Student number: 580521

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:

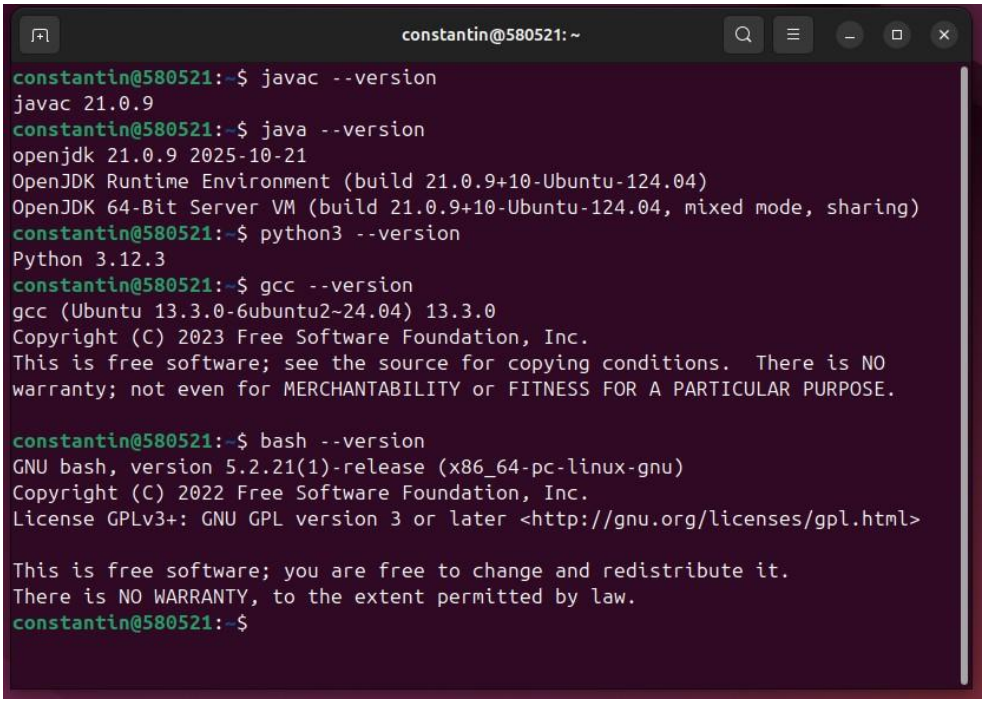
javac --version

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?

- **Java** and **C must be compiled**

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

- C

Which source code files are compiled to byte code?

- Java

Which source code files are interpreted by an interpreter?

- **Python** and **Bash**

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

- C

```
constantin@580521:~/Desktop/folder$ javac Fibonacci.java
constantin@580521:~/Desktop/folder$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.29 milliseconds
constantin@580521:~/Desktop/folder$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.40 milliseconds
constantin@580521:~/Desktop/folder$ gcc fib.c -o fib

constantin@580521:~/Desktop/folder$
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
constantin@580521:~/Desktop/folder$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.shh
constantin@580521:~/Desktop/folder$ chmod +x fib.shh
constantin@580521:~/Desktop/folder$ ./fib.shh
Fibonacci(18) = 2584
Excution time 6945 milliseconds
constantin@580521:~/Desktop/folder$
```

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?

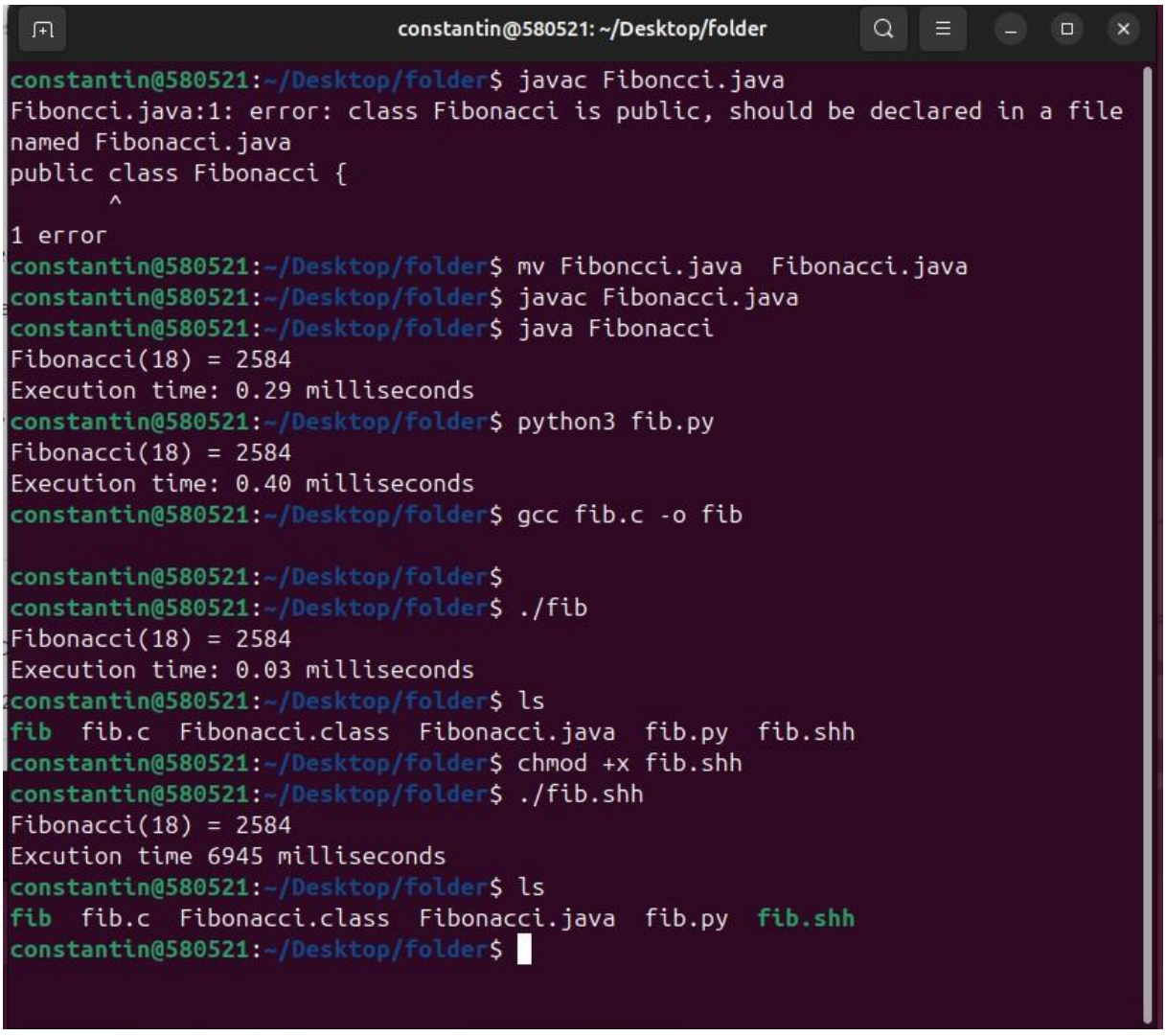
- chmod +x fib.ssh
- ./fib.ssg

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

- Fibonacci.java -> Fibonacci.class
- fib.c -> fib

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?



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.

- -O0 • -O1 • -O2
- -O3
- -Ofast

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?

```
constantin@580521:~/Desktop/folder$ gcc -O0 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
constantin@580521:~/Desktop/folder$ gcc -O1 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
constantin@580521:~/Desktop/folder$ gcc -O2 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
constantin@580521:~/Desktop/folder$ gcc -O3 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
constantin@580521:~/Desktop/folder$ gcc -Ofast fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
constantin@580521:~/Desktop/folder$
```

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.

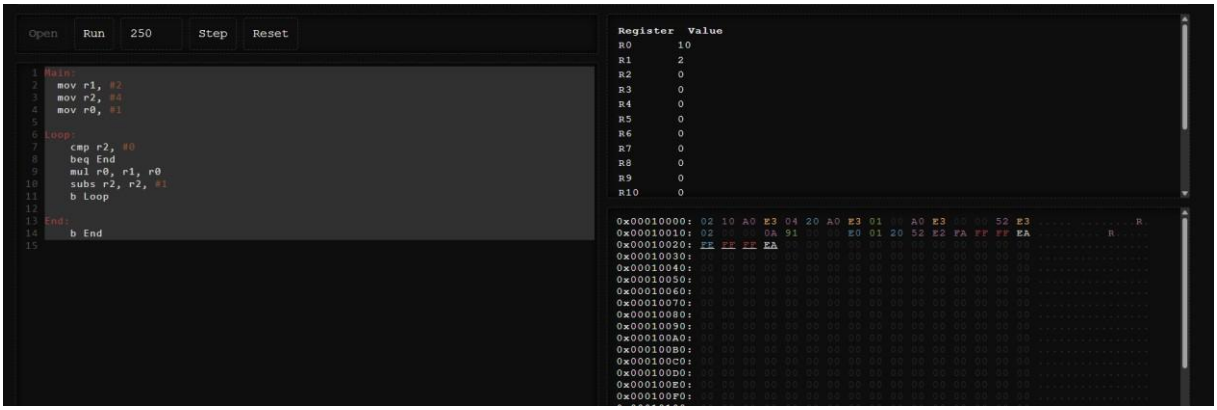
```
Main:
```

```
  mov r1, #2
mov r2, #4
mov r0, #1
Loop:
      cmp r2, #0
beq End     mul r0,
r1, r0      subs r2,
r2, #1
      b Loop

End:
      b 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**