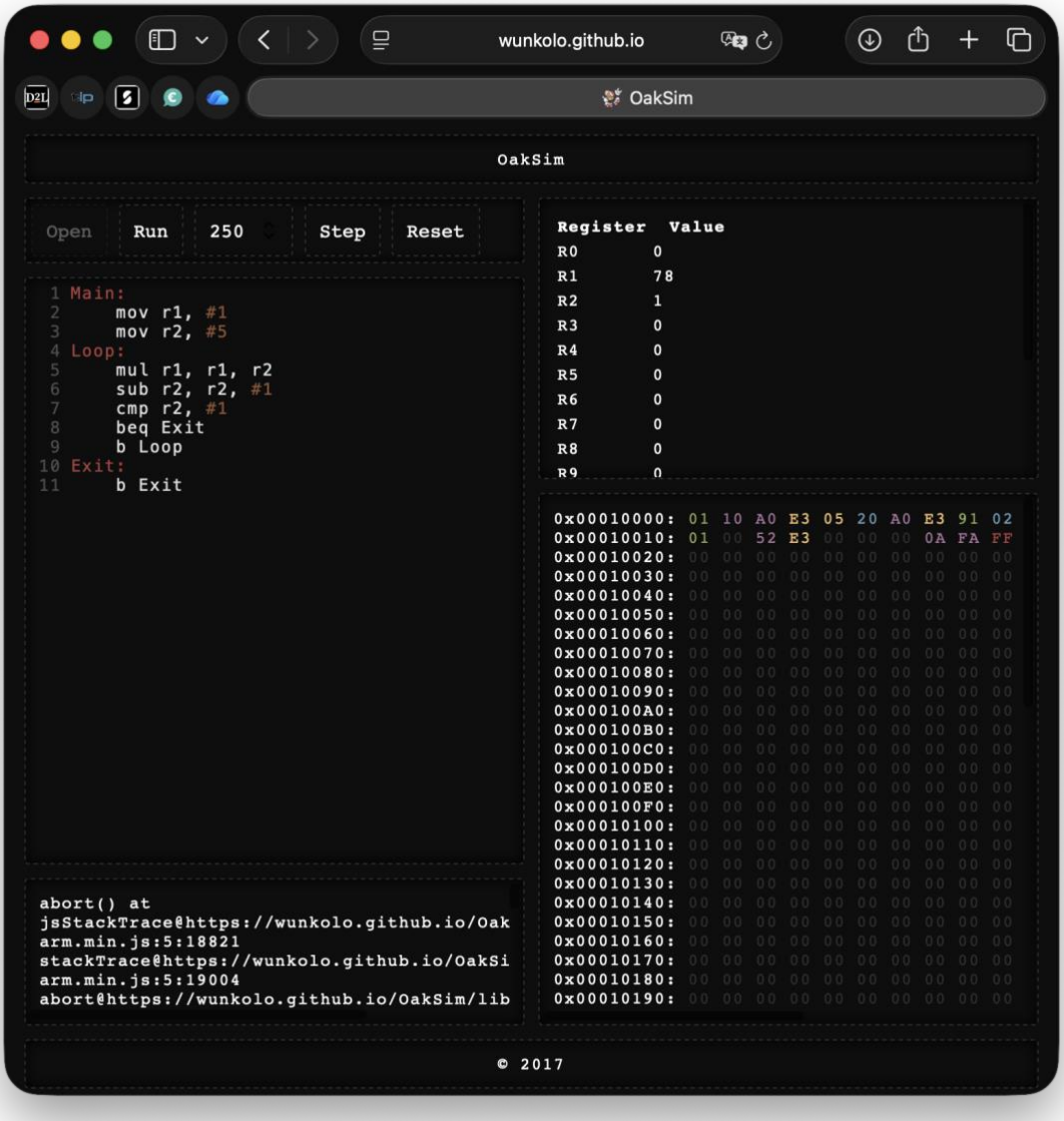
Week 4 – Software

Student number: 588406

Assignment 4.1: ARM assembly

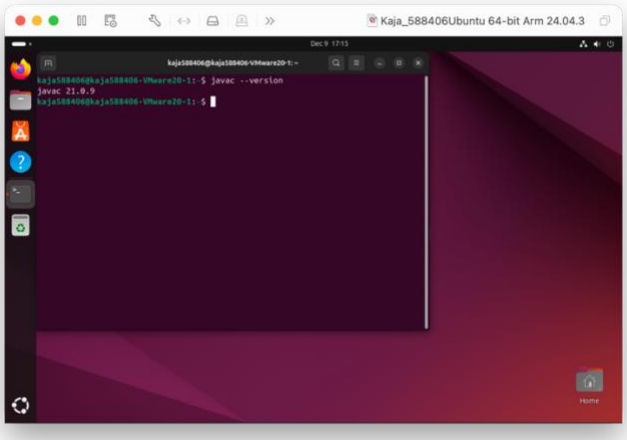
Screenshot of working assembly code of factorial calculation:



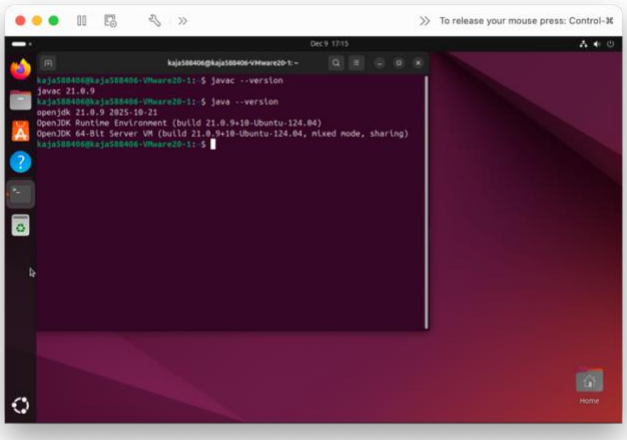
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

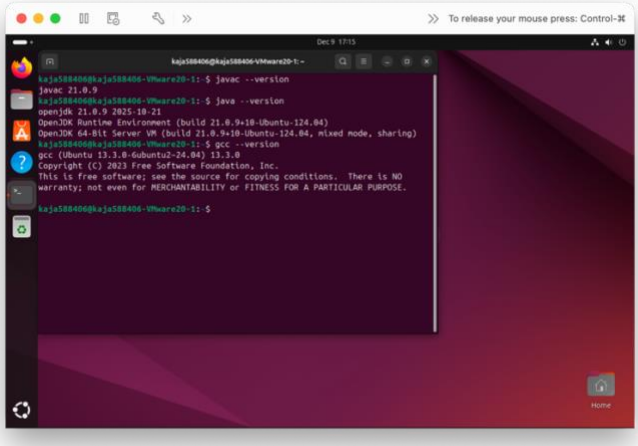
javac –version



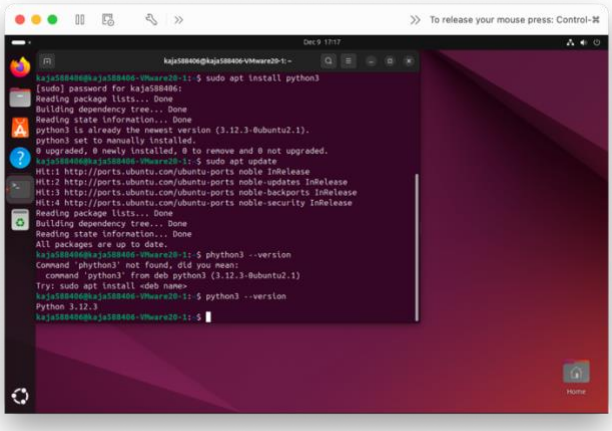
java –version



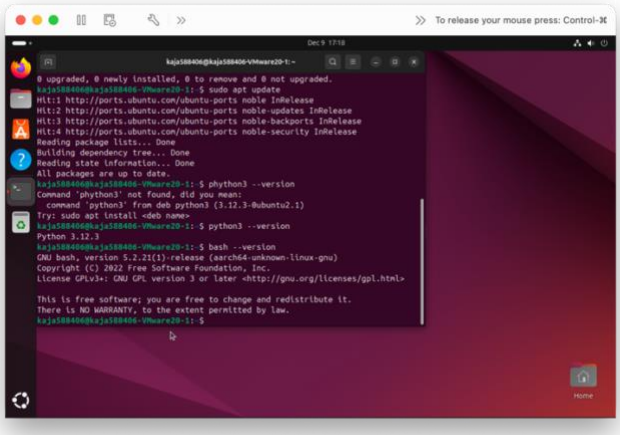
gcc –version



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 programs need to be compiled before running, while Python and Bash programs can be run directly without compilation.

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

- C source code files are compiled into machine code and can be executed directly by the processor.

Which source code files are compiled to byte code?

- Java source code files are compiled into byte code.

Which source code files are interpreted by an interpreter?

- Python and Bash source code files 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?

- C source code is expected to perform the calculation the fastest.

How do I run a Java program?

- You run a Java program by first compiling it with javac and then running it with java.

How do I run a Python program?

- You run a Python program by using the python3 command followed by the file name.

How do I run a C program?

- You run a C program by compiling it with gcc and then executing the generated file.

How do I run a Bash script?

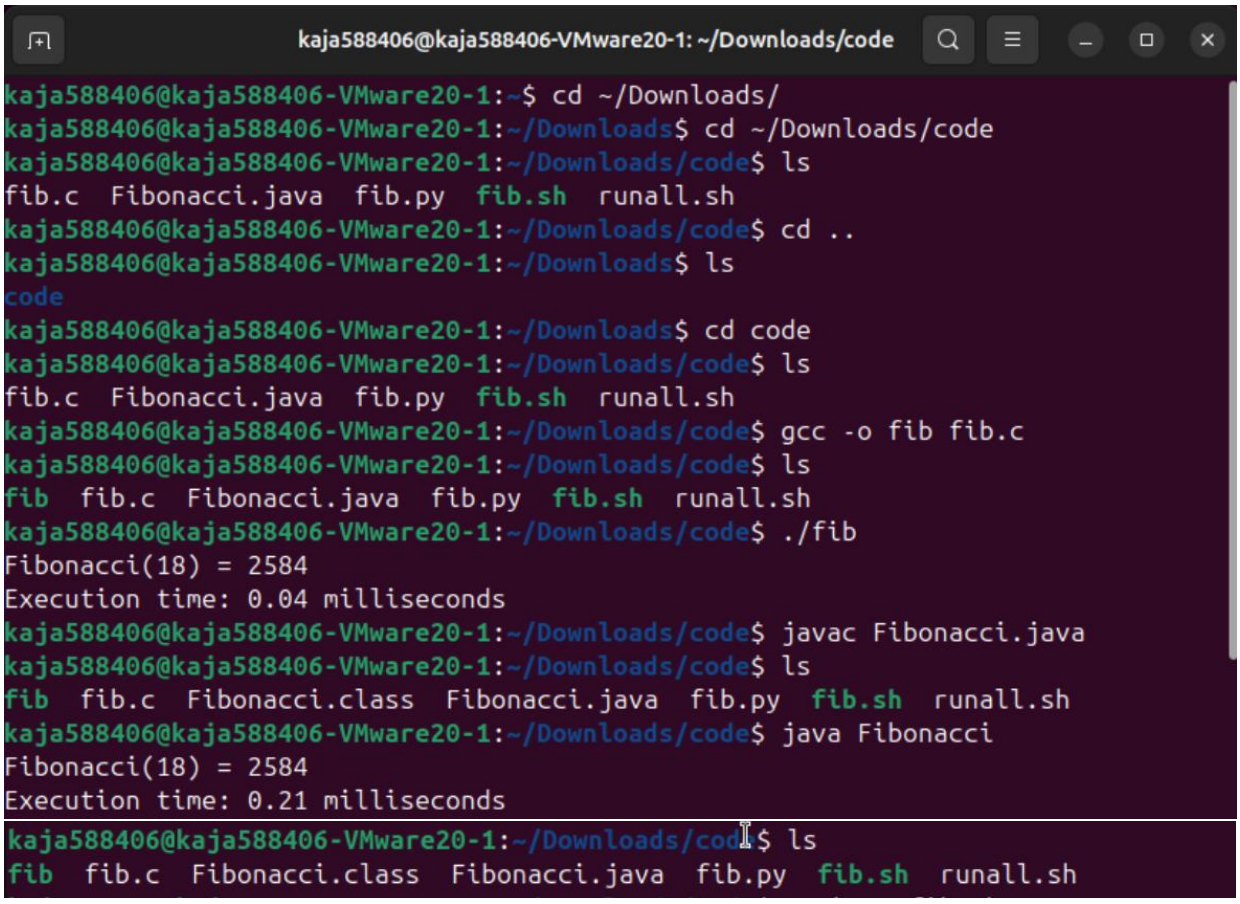
- You run a Bash script by using bash followed by the script name or by making it executable and running it directly.

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

- Yes, compiling creates a new file: Java creates a **.**class file, and C creates an executable file, while Python and Bash do not create a new file when run.

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?



```
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.56 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ chmod +x fib.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ chmod +x runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 1972 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo chmod a+x fib.sh
[sudo] password for kaja588406:
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 1946 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$
```

The C program is the fastes, because it runs directly as machine cos

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.

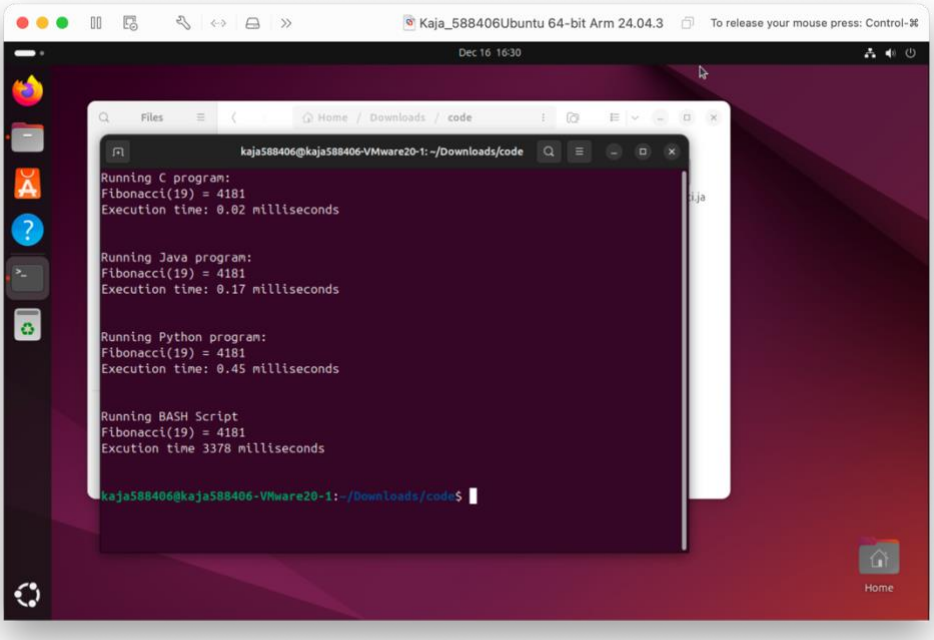
b) Compile **fib.c** again with the optimization parameters

```
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo gcc -O2 fib.c -o fib
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$
```

c) Run the newly compiled program. Is it true that it now performs the calculation faster?
Yes the calculation was 0.27 milliseconds faster after I optimised it.

```
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo gcc -O fib.c -o fib
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.26 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo gcc -O3 fib.c -o fib
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 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.



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.

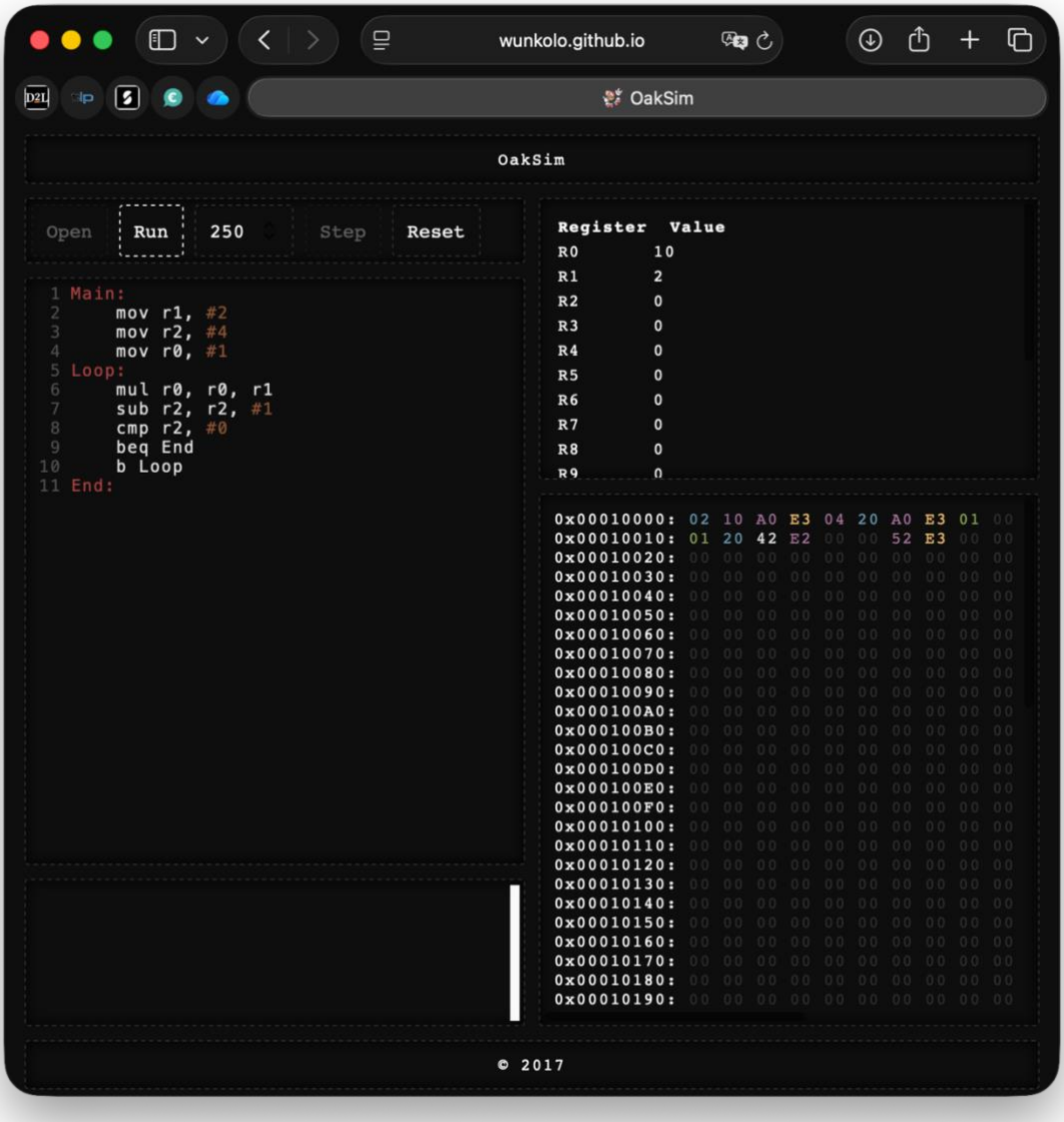
```
Main:
mov r1, #2
mov r2, #4
mov r0, #1

Loop:
mul r0, r0, r1
subs r2, r2, #1
cmp r2, #0
beq End
b 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**