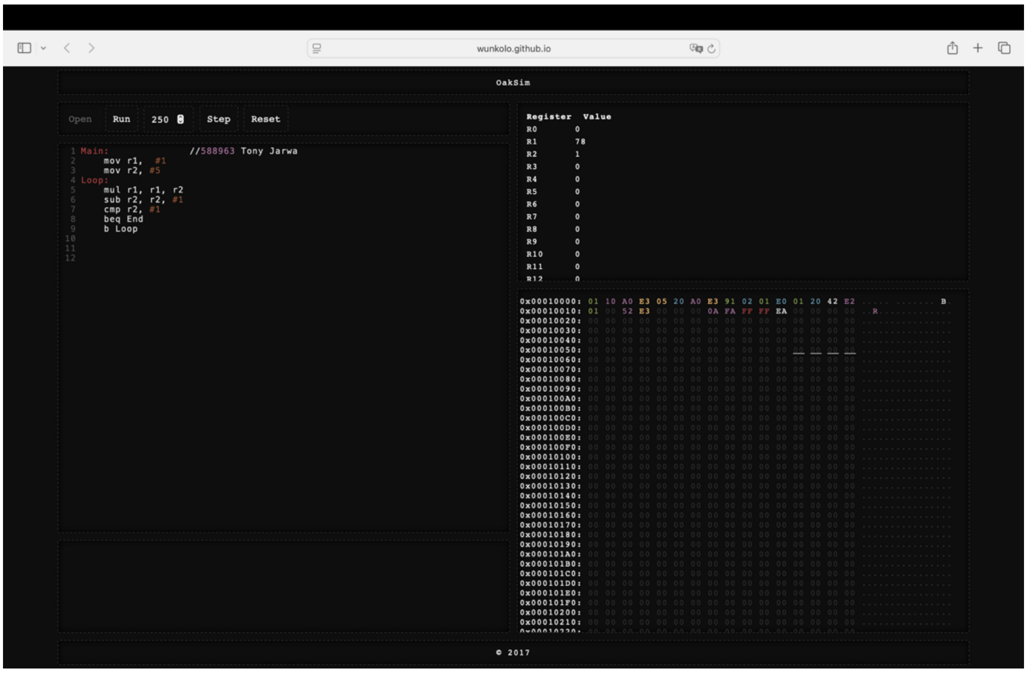
Template Week 4 – Software

Student number: 588963

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

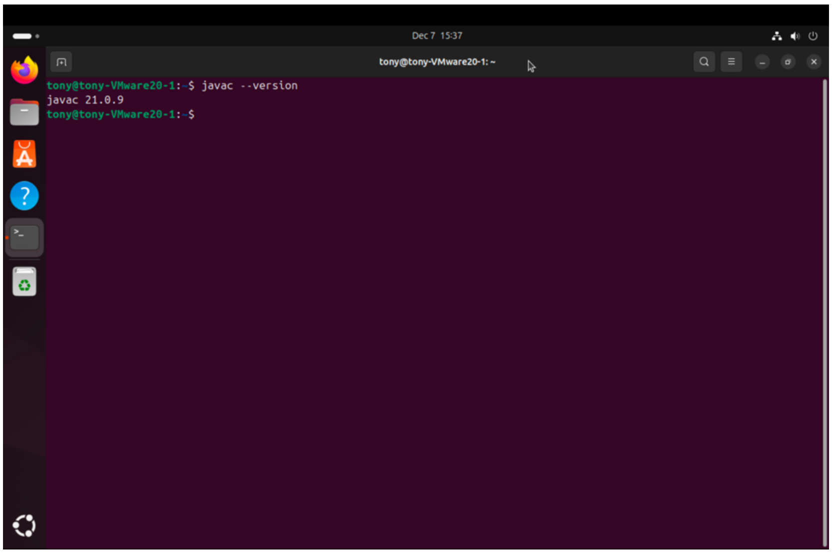


78 in Hex is 120 in Decimal.

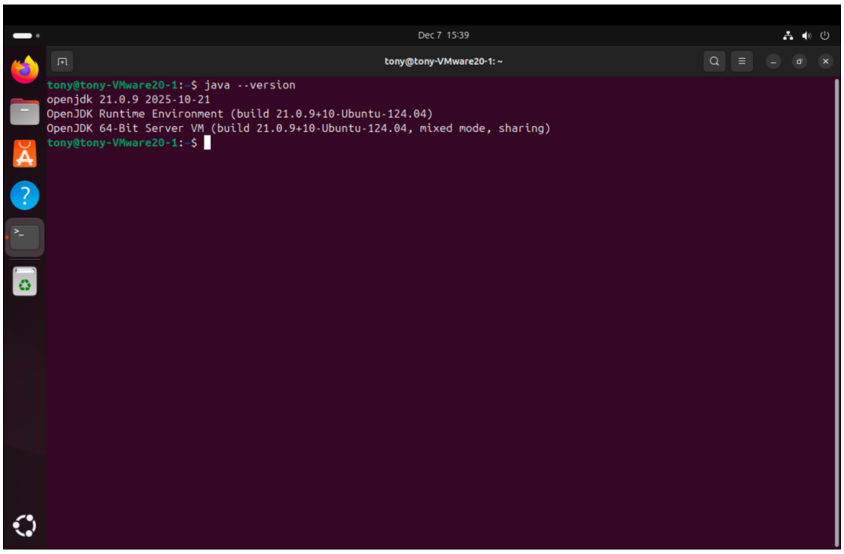
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

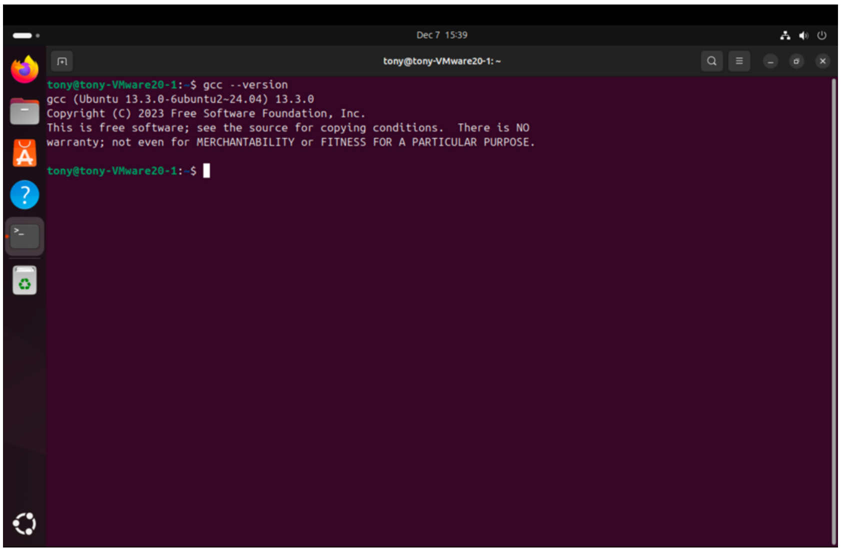
javac –version



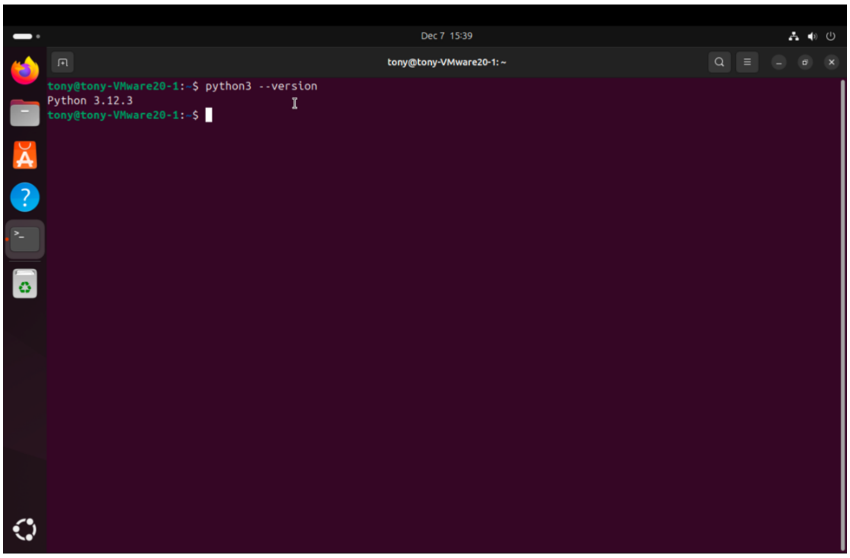
java –version



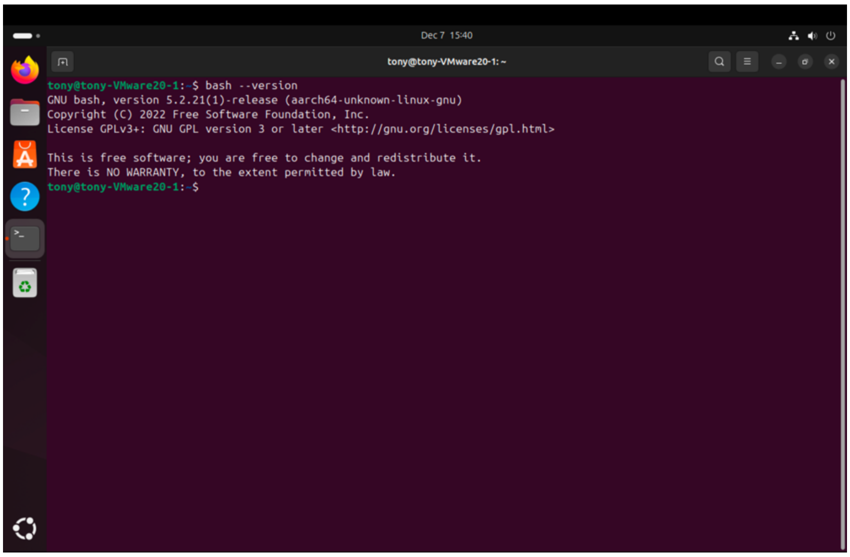
gcc –version



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?

The fib.c (C program) is expected to be the fastest one because it is compiled directly into machine code.

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 a+x fib.sh

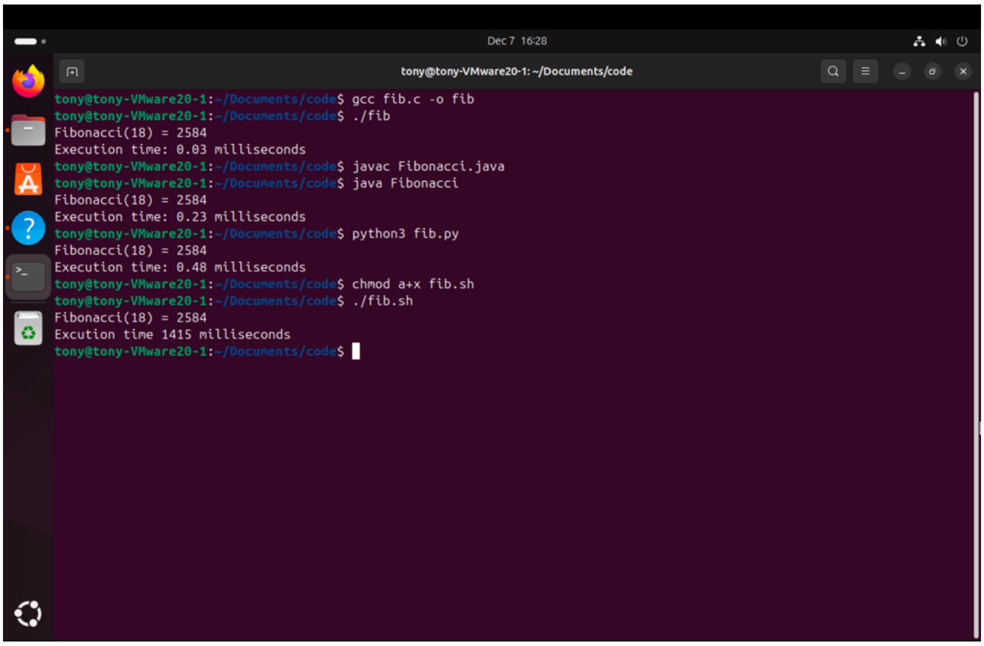
./fib.sh

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

Yes, by Fibonacci.java a new file (Fibonacci.class), and 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?

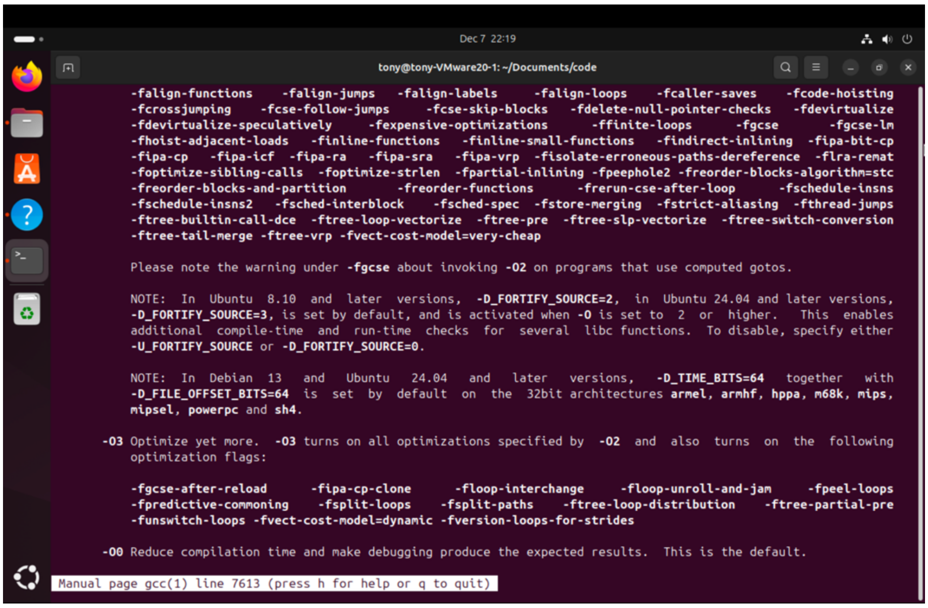


The C program (fib.c) is the fastest (0,03 milliseconds).

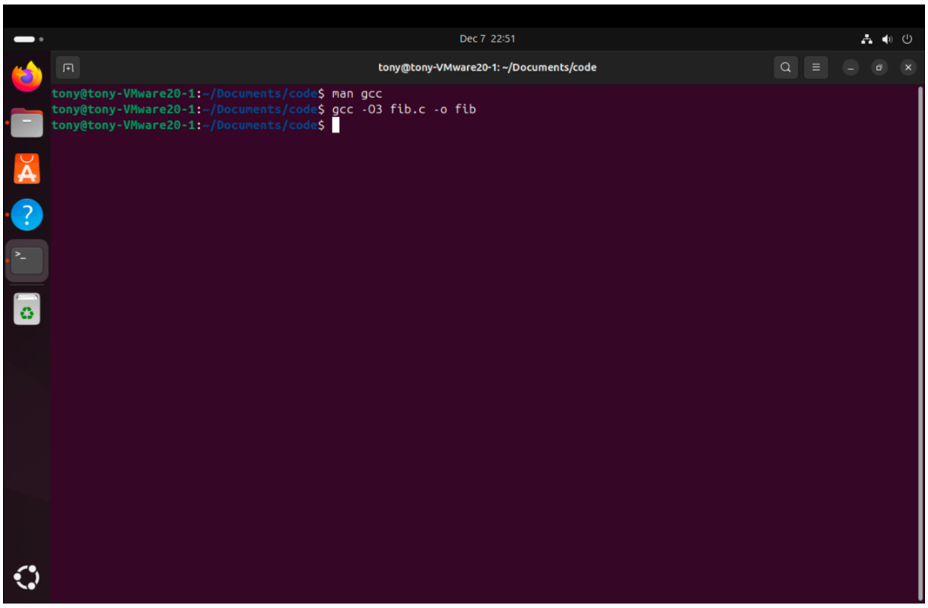
Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

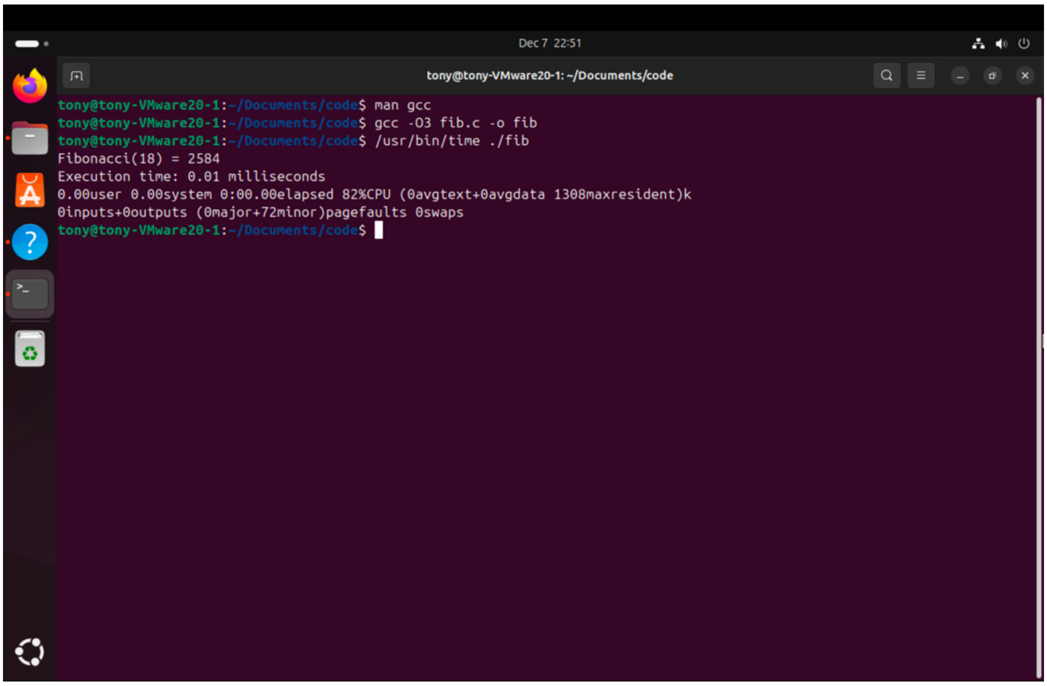
a) Figure out which parameters you need to pass to **the gcc** compiler so that the compiler performs a number of optimizations that will ensure that the compiled source code will run faster. **Tip!** The parameters are usually a letter followed by a number. Also read **page 191** of your book but find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



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

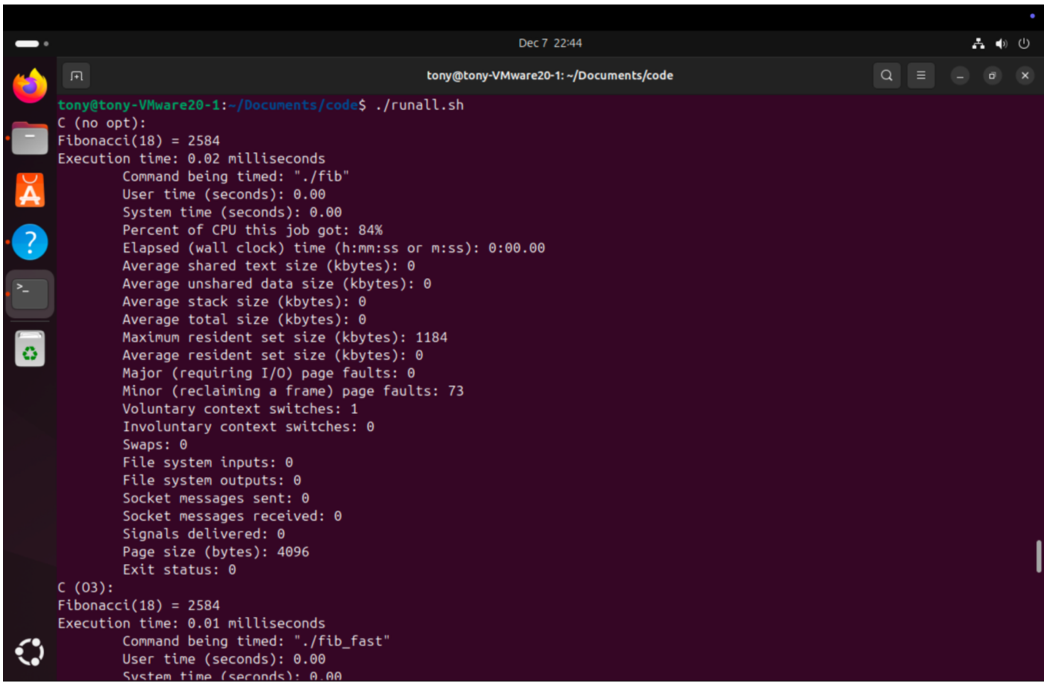


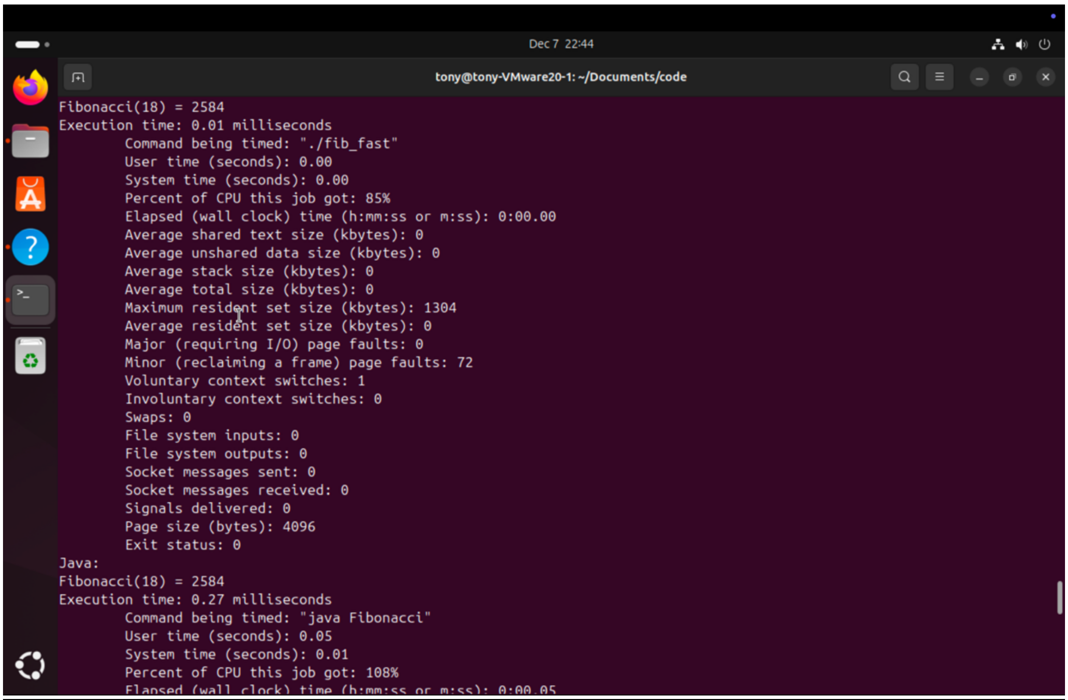
c) Run the newly compiled program. Is it true that it now performs the calculation faster?

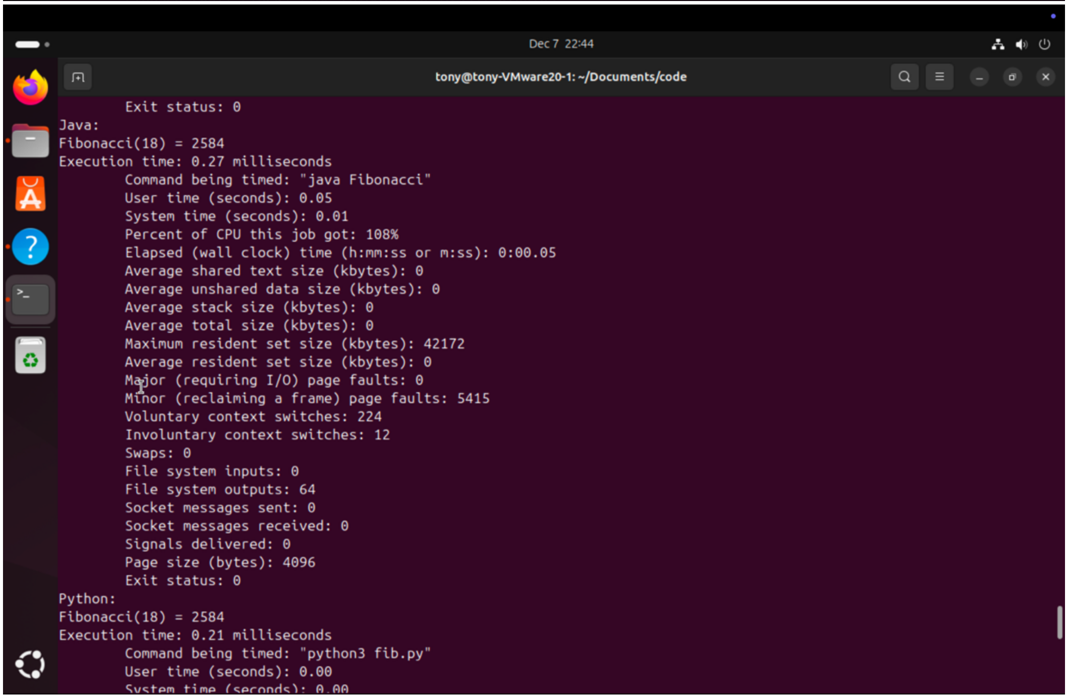


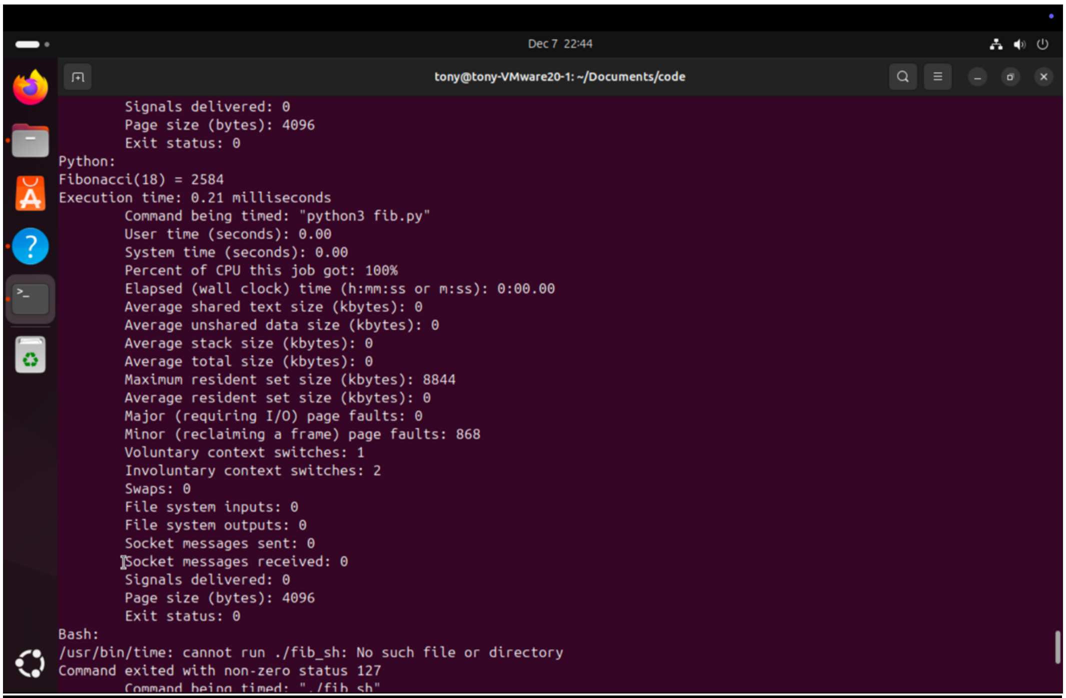
It is sure faster (0,01 milliseconds).

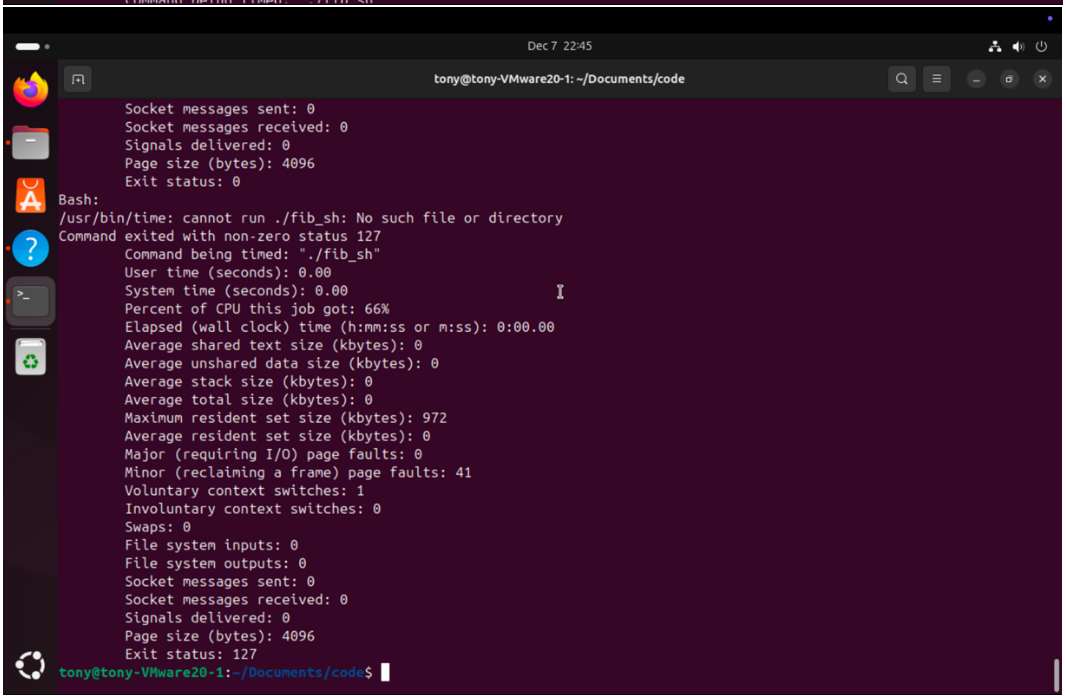
d) Edit the file **runall.sh**, so you can perform all four calculations in a row using this Bash script. So, the (compiled/interpreted) C, Java, Python and Bash versions of Fibonacci one after the other.




Dec 7 22:44
tony@tony-VMware20-1: ~/Documents/code
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
Command being timed: "./fib_fast"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 85%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1304
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 72
Voluntary context switches: 1
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
Java:
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
Command being timed: "java Fibonacci"
User time (seconds): 0.05
System time (seconds): 0.01
Percent of CPU this job got: 108%


Dec 7 22:44
tony@tony-VMware20-1: ~/Documents/code
Exit status: 0
Java:
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
Command being timed: "java Fibonacci"
User time (seconds): 0.05
System time (seconds): 0.01
Percent of CPU this job got: 108%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.05
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 42172
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 5415
Voluntary context switches: 224
Involuntary context switches: 12
Swaps: 0
File system inputs: 0
File system outputs: 64
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
Python:
Fibonacci(18) = 2584
Execution time: 0.21 milliseconds
Command being timed: "python3 fib.py"
User time (seconds): 0.00





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.

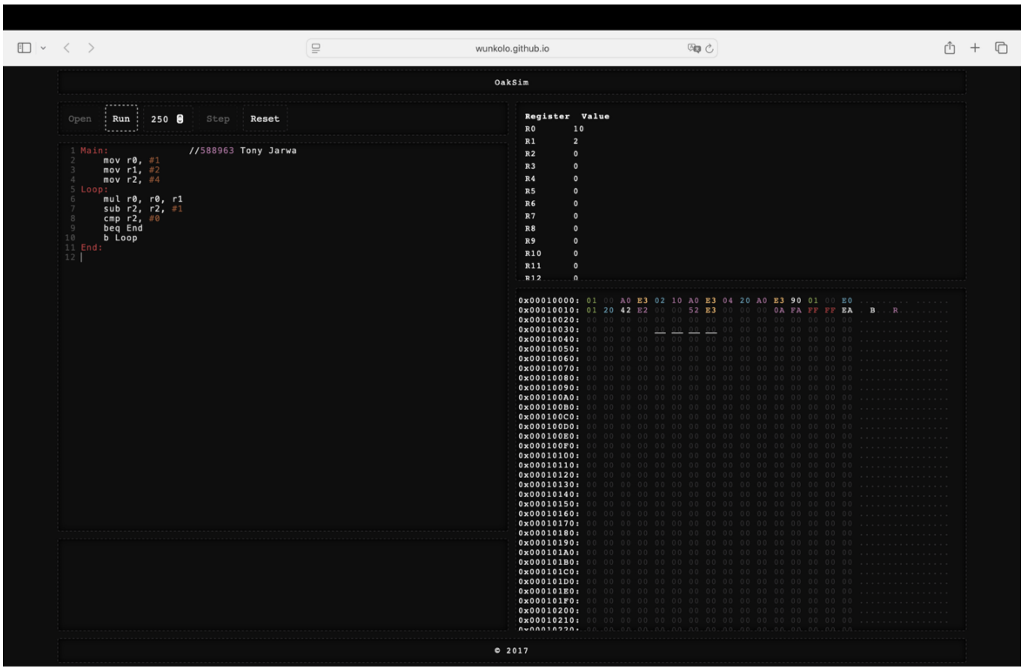
```
Main:

mov r1, #2

mov r2, #4
```

Loop:

End:

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

Screenshot of the completed code here.



10 in Hex is 16 in Decimal.

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