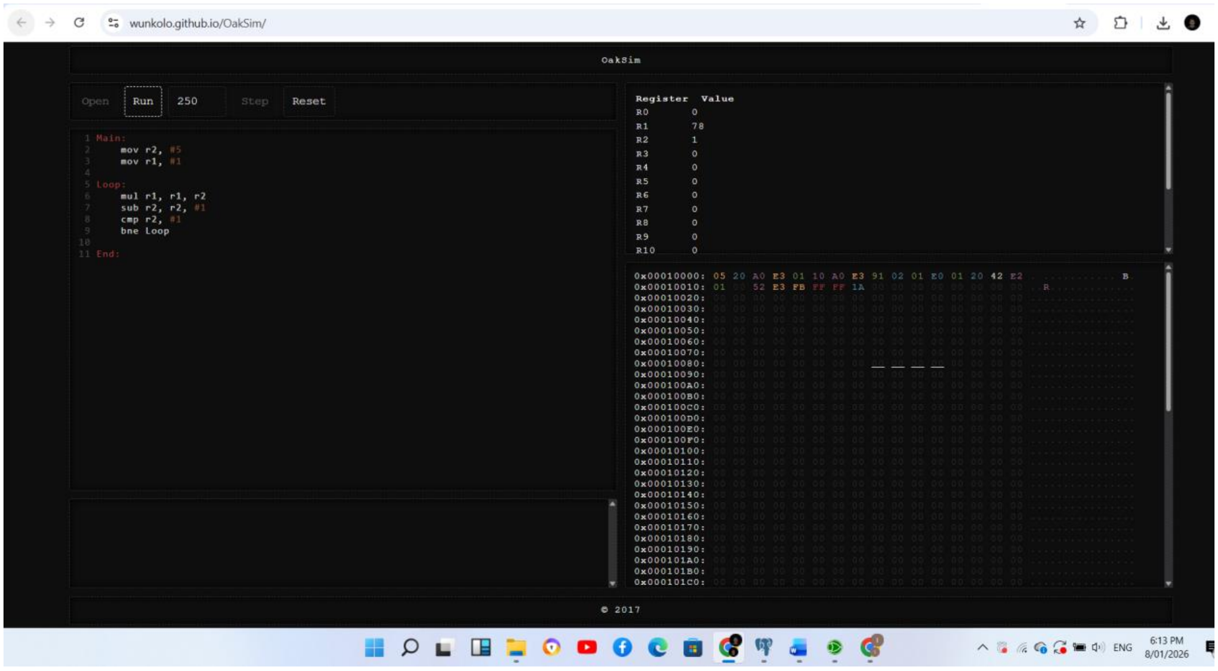
Template Week 4 – Software

Student number:

Ebrahim 577534

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
    mov r2, #5
    mov r1, #1

Loop:
    mul r1, r1, r2
    sub r2, r2, #1
    cmp r2, #1
    bne Loop

End:
```

```
Register  Value
R0        0
R1        78
R2        1
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
```

```
0x00010000: 05 20 A0 E3 01 10 A0 E3 91 02 01 E0 01 20 42 E2 ...........B.
0x00010010: 01 00 52 E3 FB FF FF 1A 00 00 00 00 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 ................
0x000100D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000101A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000101B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000101C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

6:13 PM
8/01/2026

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

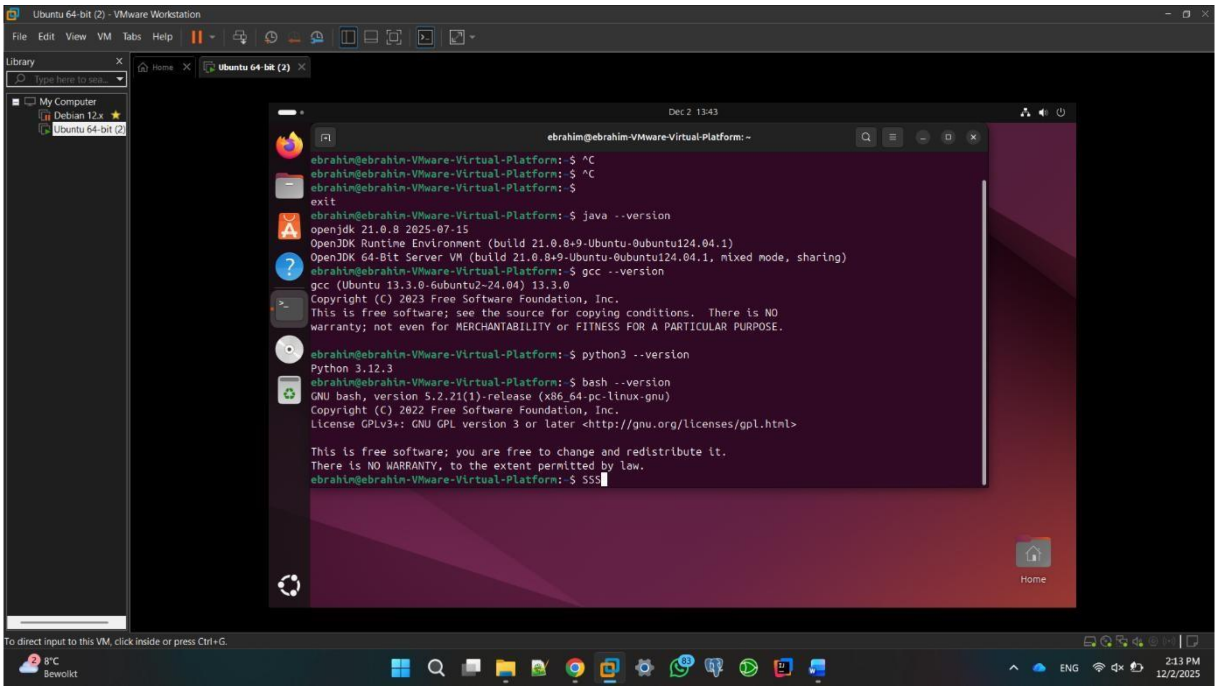
javac –version : javac 21.0.9

java –version : openjdk 21.0.9 2024-10-15

gcc –version :   gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0

python3 –version   :   Python 3.12.3

bash –version : GNU bash, version 5.2.21


Ubuntu 64-bit (2) - VMware Workstation
File Edit View VM Tabs Help
Library
My Computer
Debian 12.x
Ubuntu 64-bit (2)
Home
Ubuntu 64-bit (2)
Dec 2 13:43
ebrahim@ebrahim-VMware-Virtual-Platform: ~
Home
To direct input to this VM, click inside or press Ctrl+G.
8°C
Bewolkt
ENG
2:13 PM
12/2/2025

```
ebrahim@ebrahim-VMware-Virtual-Platform:~$ ^C
ebrahim@ebrahim-VMware-Virtual-Platform:~$ ^C
ebrahim@ebrahim-VMware-Virtual-Platform:~$
exit
ebrahim@ebrahim-VMware-Virtual-Platform:~$ java --version
openjdk 21.0.8 2025-07-15
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)
ebrahim@ebrahim-VMware-Virtual-Platform:~$ 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.

ebrahim@ebrahim-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
ebrahim@ebrahim-VMware-Virtual-Platform:~$ 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.
ebrahim@ebrahim-VMware-Virtual-Platform:~$ SSS
```

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 → fib

Which source code files are compiled to byte code? Fibonacci.java → Fibonacci.class

Which source code files are interpreted by an interpreter?

fib.py (Python)
fib.sh (Bash)

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

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.sh
./fib.sh
```

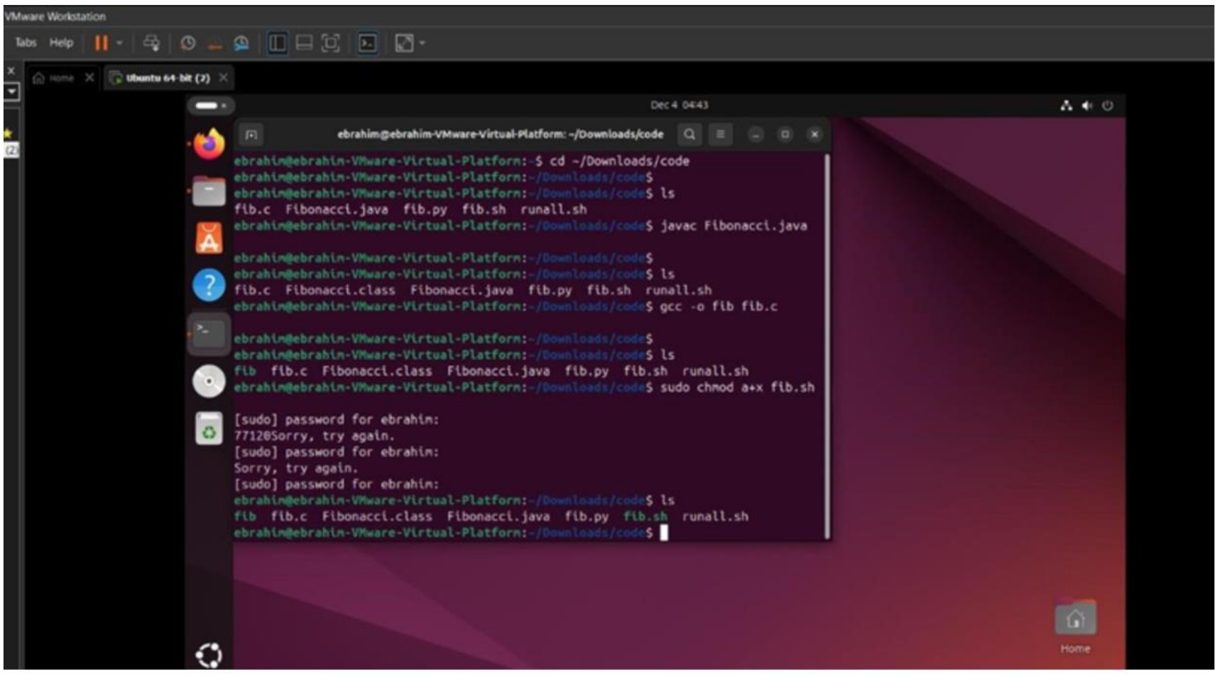
If I compile the above source code, will a new file be created? If so, which file?

Fibonacci.class
fib (native executable)

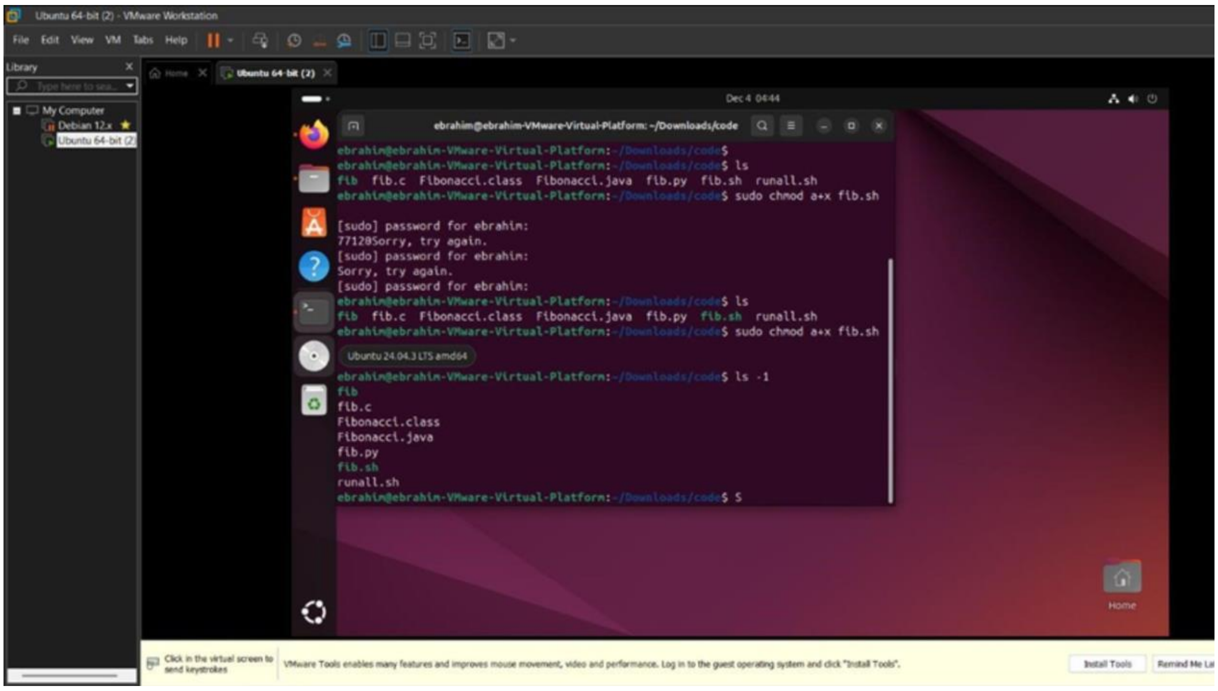
Take relevant screenshots of the following commands:

- 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`) is the fastest



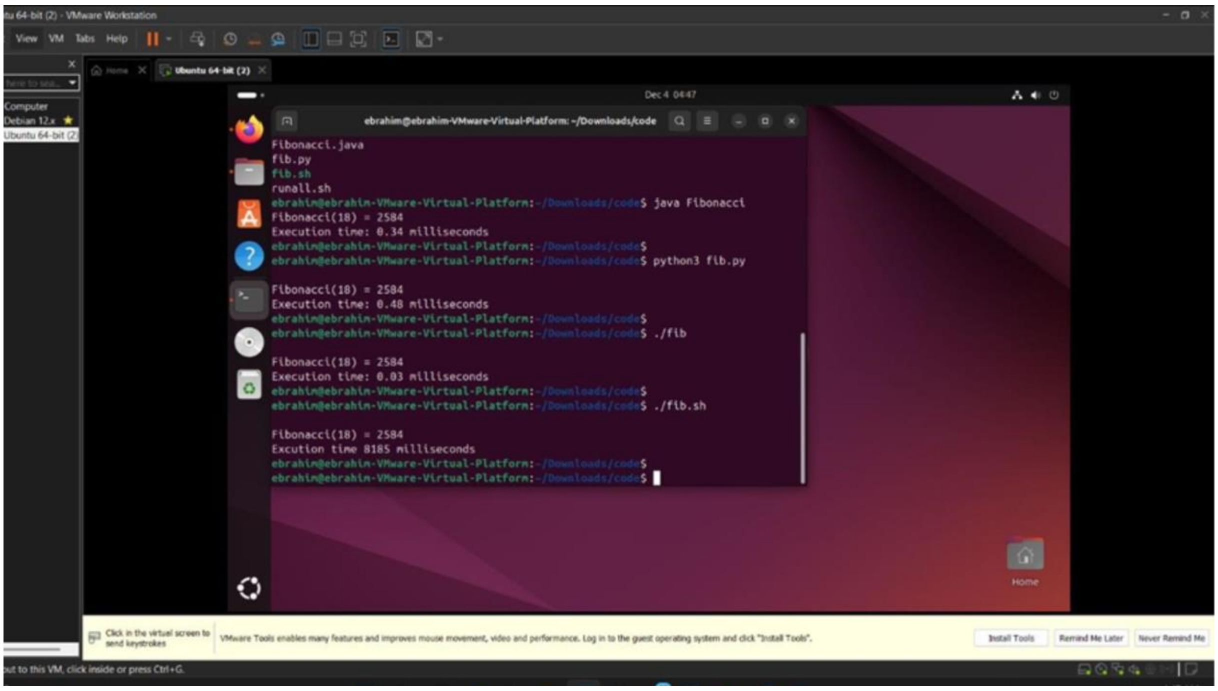
```
ebrahim@ebrahim-VMware-Virtual-Platform: ~/Downloads/code

ebrahim@ebrahim-VMware-Virtual-Platform:~$ cd ~/Downloads/code
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java

ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ gcc -o fib fib.c

ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x fib.sh

[sudo] password for ebrahim:
77120Sorry, try again.
[sudo] password for ebrahim:
Sorry, try again.
[sudo] password for ebrahim:
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
```



```
ebrahim@ebrahim-VMware-Virtual-Platform: ~/Downloads/code

ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x fib.sh

[sudo] password for ebrahim:
77120Sorry, try again.
[sudo] password for ebrahim:
Sorry, try again.
[sudo] password for ebrahim:
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x fib.sh
 Ubuntu 24.04.3 LTS amd64
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls -1
fib
fib.c
Fibonacci.class
Fibonacci.java
fib.py
fib.sh
runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ $
```

Click in the virtual screen to send keystrokes

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".



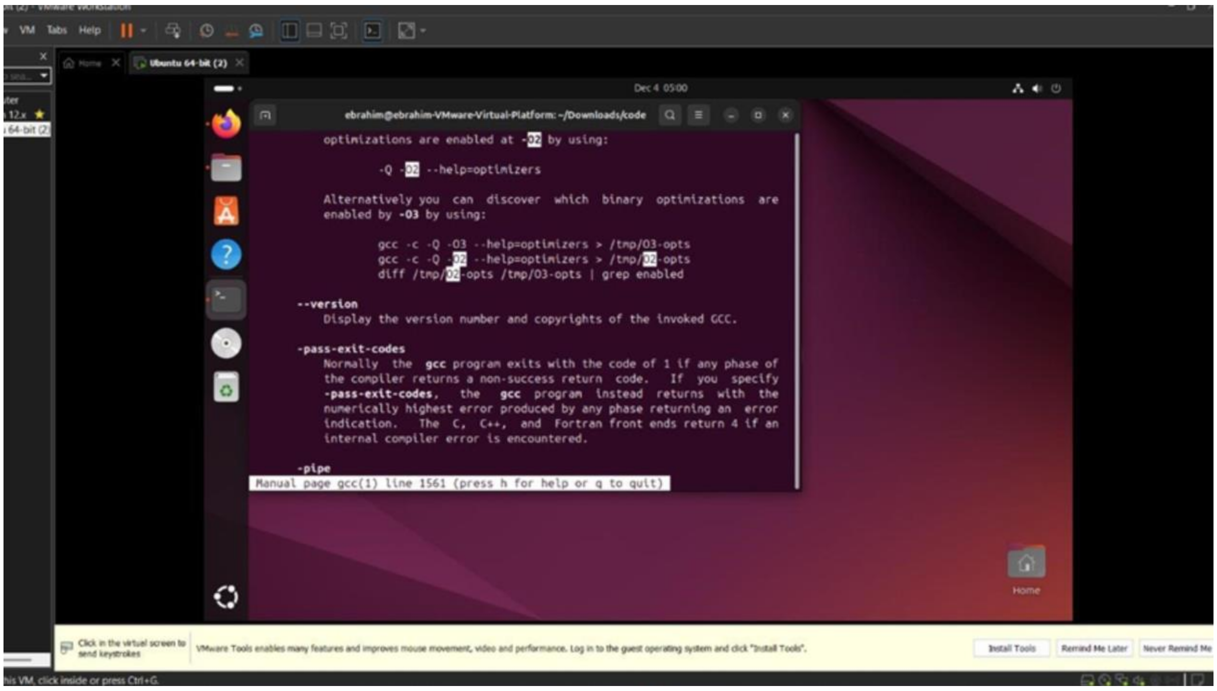
```
Fibonacci.java
fib.py
fib.sh
runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.34 milliseconds
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py

Fibonacci(18) = 2584
Execution time: 0.48 milliseconds
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ./fib

Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ./fib.sh

Fibonacci(18) = 2584
Excution time 8185 milliseconds
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
```

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.

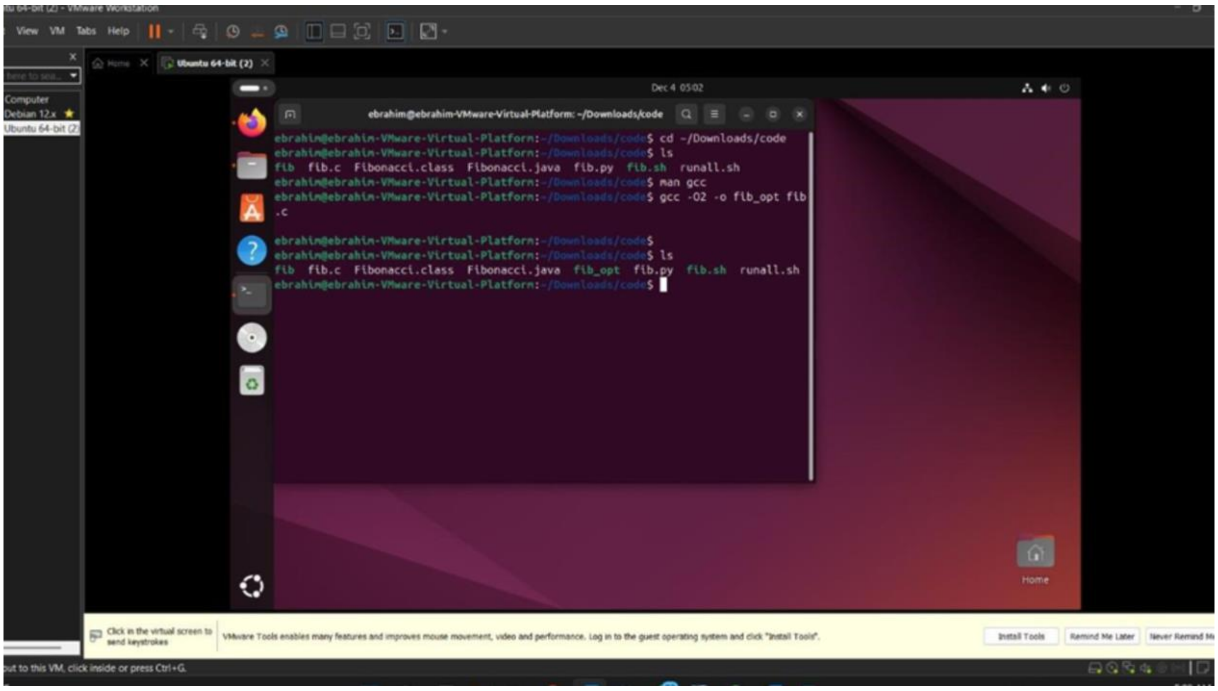
   Programs written in C usually run faster because the compiler translates them straight into machine code that the processor can execute directly. Java first compiles to bytecode, which must be executed by the JVM, while Python and Bash are interpreted step by step at runtime. These extra execution layers introduce additional overhead, so compared to them, a C program is generally the most efficient in terms of speed.



```
ebrahim@ebrahim-VMware-Virtual-Platform: ~/Downloads/code
           optimizations are enabled at -O2 by using:

                   -Q -O2 --help=optimizers

           Alternatively you  can  discover  which  binary  optimizations  are
           enabled by -O3 by using:

                   gcc -c -Q -O3 --help=optimizers > /tmp/O3-opts
                   gcc -c -Q -O2 --help=optimizers > /tmp/O2-opts
                   diff /tmp/O2-opts /tmp/O3-opts | grep enabled

       --version
           Display the version number and copyrights of the invoked GCC.

       -pass-exit-codes
           Normally  the  gcc program exits with the code of 1 if any phase of
           the compiler returns a non-success return  code.   If  you  specify
           -pass-exit-codes,   the  gcc  program  instead  returns  with  the
           numerically highest error produced by any phase returning an  error
           indication.   The  C,  C++,  and  Fortran front ends return 4 if an
           internal compiler error is encountered.

       -pipe
 Manual page gcc(1) line 1561 (press h for help or q to quit)
```

written as a capital letter **O** followed by a number. Examples are **-O1**, **-O2** and **-O3**. The option **-O2** enables a wide range of optimizations that usually make the compiled program faster without changing its behaviour. The option **-O3** enables even more aggressive optimizations, focusing on speed, but it may increase compilation time and code size. In this assignment I used **-O2** as an optimization level for fib.c

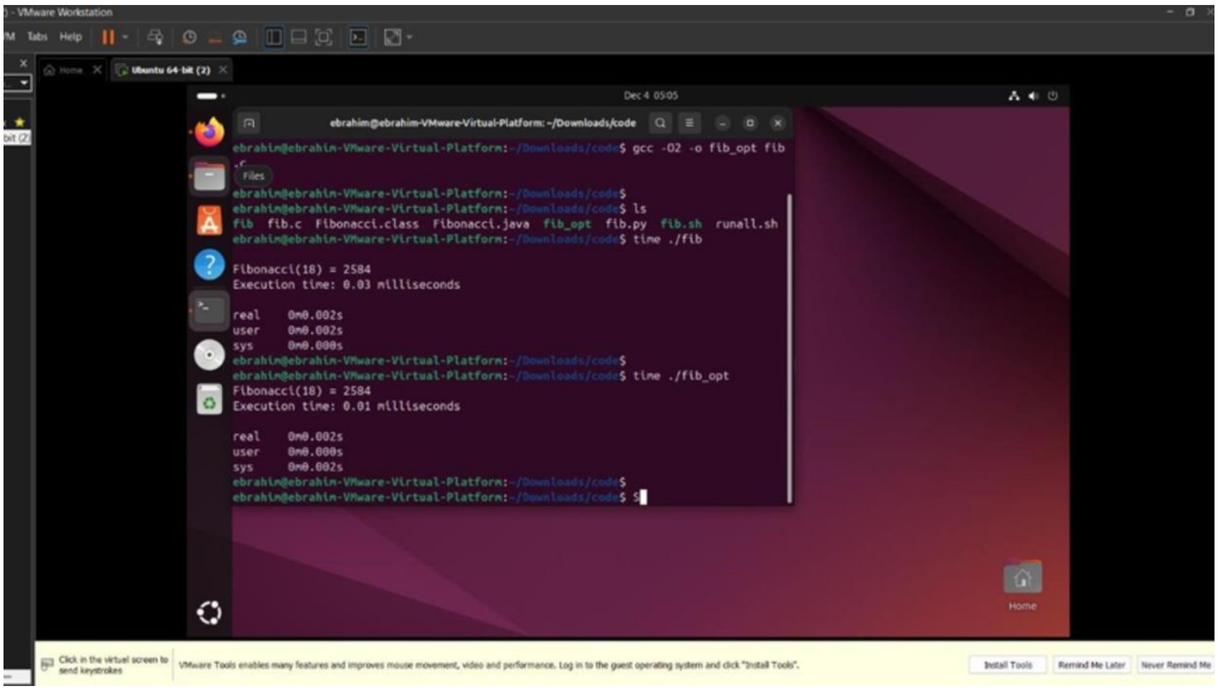
b) Compile fib.c again with the optimization parameters



```
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ cd ~/Downloads/code
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ man gcc
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ gcc -O2 -o fib_opt fib
.c

ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib_opt  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ 
```

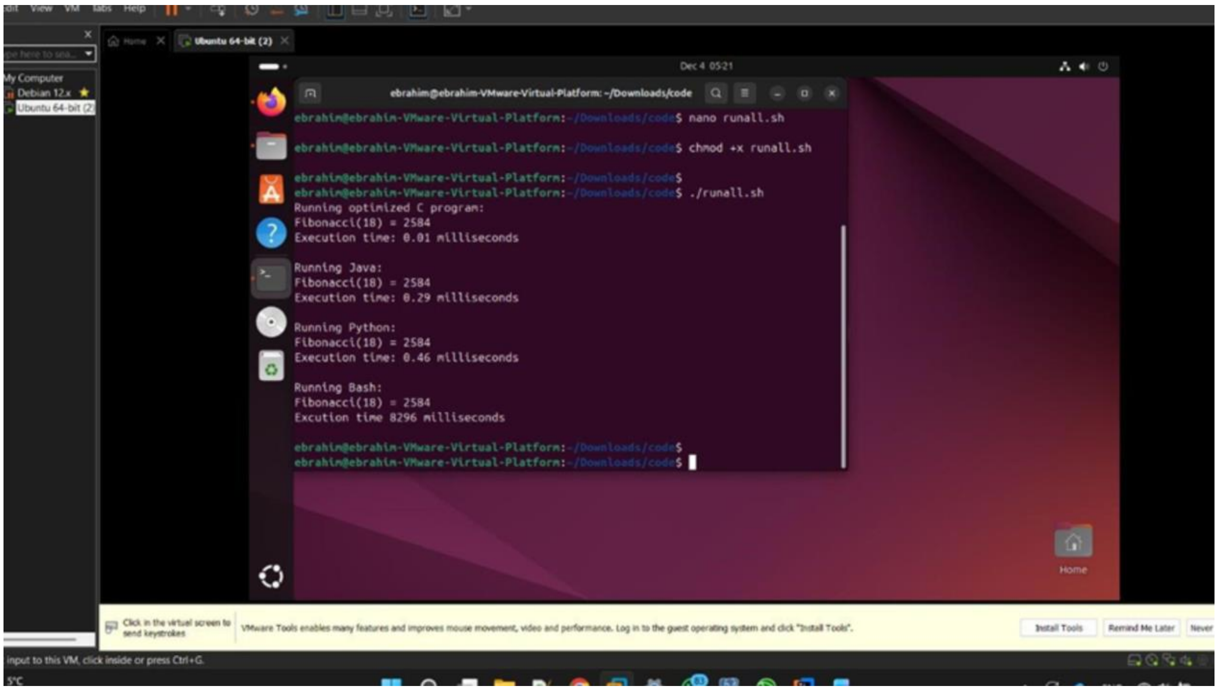
c) Run the newly compiled program. Is it true that it now performs the calculation faster?

Yes, after running the newly compiled program, it generally performs the calculation a bit faster because the compiler optimizations improve how the program executes, even if the difference is small for a simple program.



```
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ gcc -O2 -o fib_opt fib
.c
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib_opt  fib.py  fib.sh  runall.sh
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ time ./fib

Fibonacci(18) = 2584
Execution time: 0.03 milliseconds

real    0m0.002s
user    0m0.002s
sys     0m0.000s
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.000s
sys     0m0.002s
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ $
```

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.



```
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ nano runall.sh

ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ chmod +x runall.sh

ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$ ./runall.sh
Running optimized C program:
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

Running Java:
Fibonacci(18) = 2584
Execution time: 0.29 milliseconds

Running Python:
Fibonacci(18) = 2584
Execution time: 0.46 milliseconds

Running Bash:
Fibonacci(18) = 2584
Excution time 8296 milliseconds

ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
ebrahim@ebrahim-VMware-Virtual-Platform:~/Downloads/code$
```

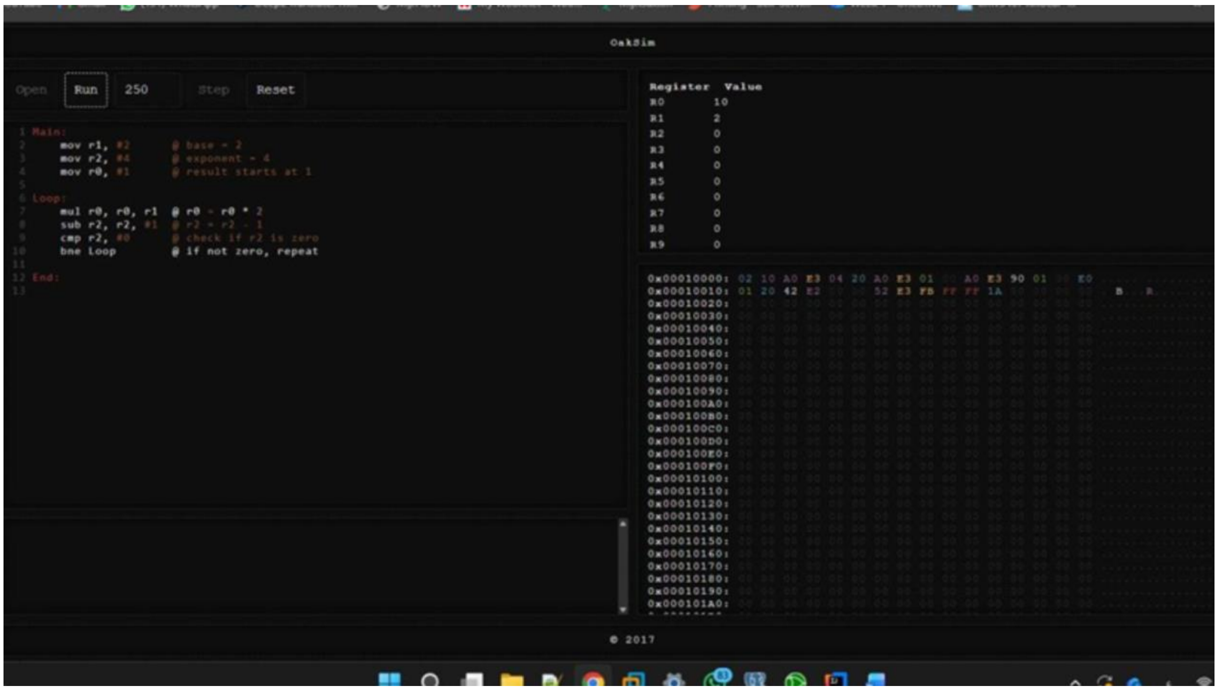
Assignment 4.5: More ARM Assembly

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, #
Loop:
  mul r0, r0, r1
sub r2, r2, #1
cmp r2, #0
bne 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 starts at 1

Loop:
    mul r0, r0, r1  @ r0 = r0 * 2
    sub r2, r2, #1  @ r2 = r2 - 1
    cmp r2, #0        @ check if r2 is zero
    bne Loop          @ if not zero, repeat

End:
```

```
Register  Value
R0        10
R1        2
R2        0
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
```

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