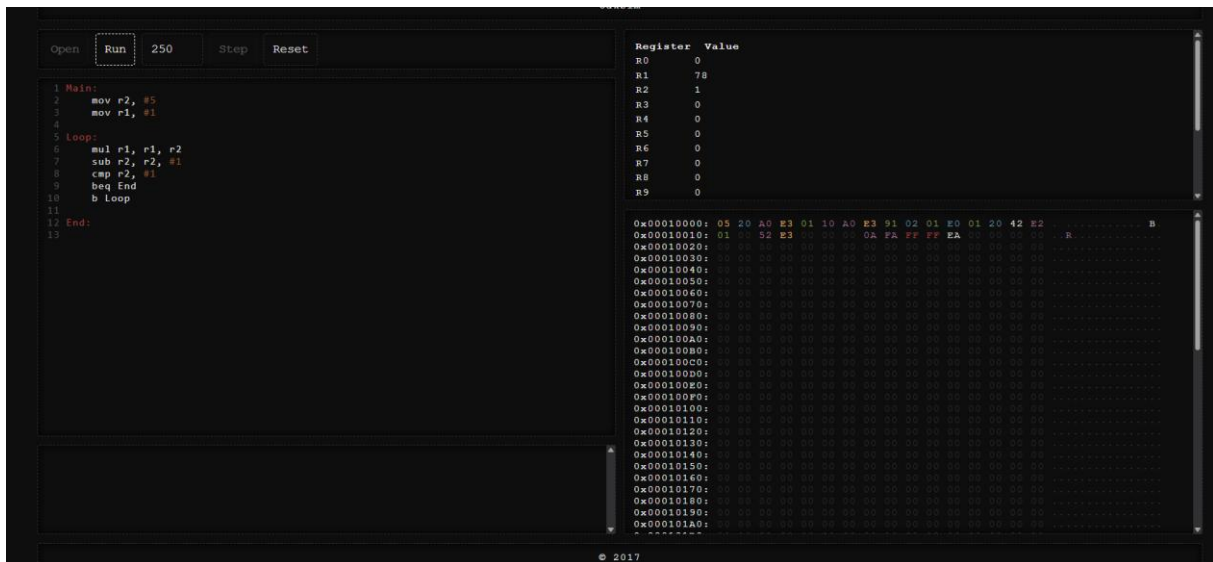


# Week 4 – Software

Student number: 589932

## Assignment 4.1: ARM assembly

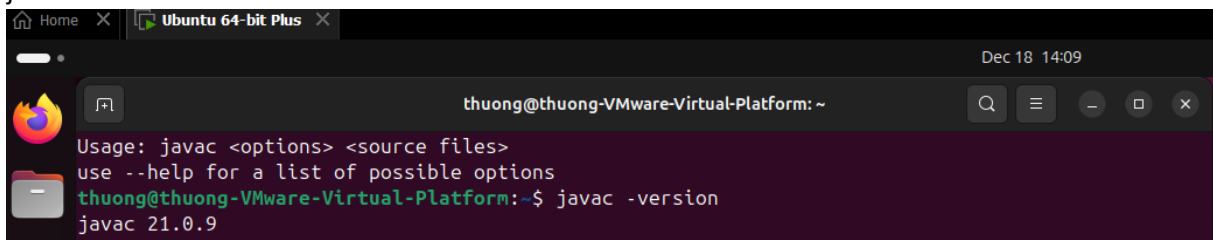
Screenshot of working assembly code of factorial calculation:



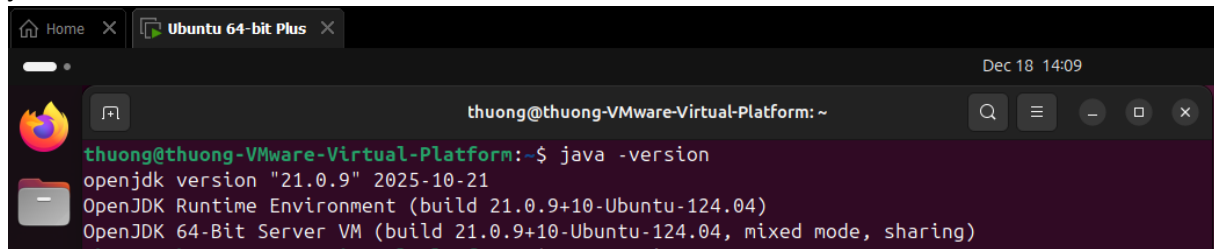
## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac -version

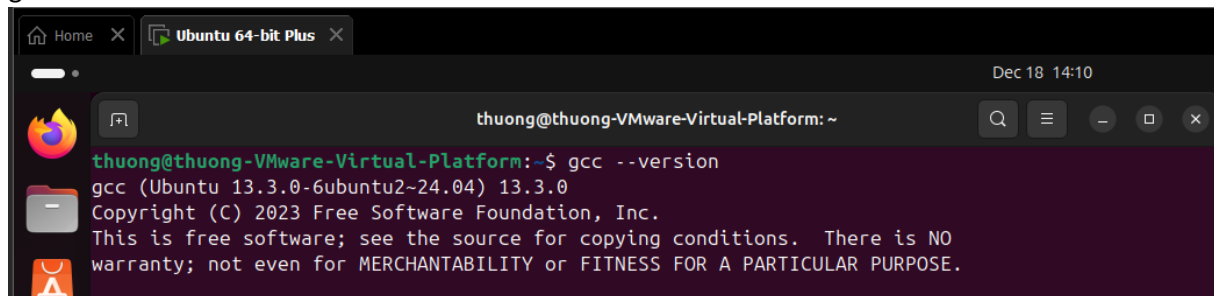


java -version

A terminal window titled 'Ubuntu 64-bit Plus' with the user 'thuong@thuong-VMware-Virtual-Platform: ~'. The command 'java -version' has been executed, resulting in the following output: 'openjdk version "21.0.9" 2025-10-21', 'OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)', and 'OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)'.

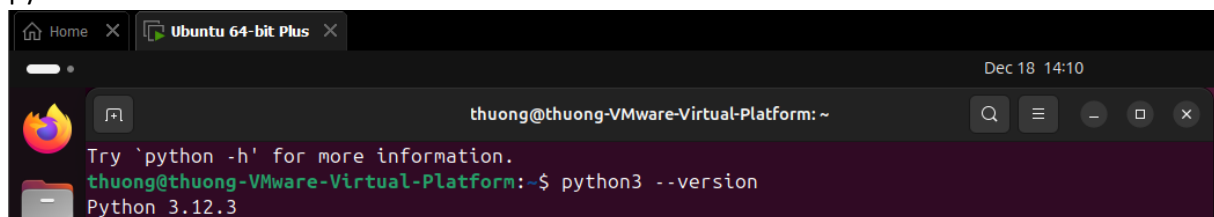
```
thuong@thuong-VMware-Virtual-Platform:~$ java -version
openjdk version "21.0.9" 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
```

gcc --version

A terminal window titled 'Ubuntu 64-bit Plus' with the user 'thuong@thuong-VMware-Virtual-Platform: ~'. The command 'gcc --version' has been executed, resulting in the following output: 'gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0', 'Copyright (C) 2023 Free Software Foundation, Inc.', and a license notice: 'This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.'.

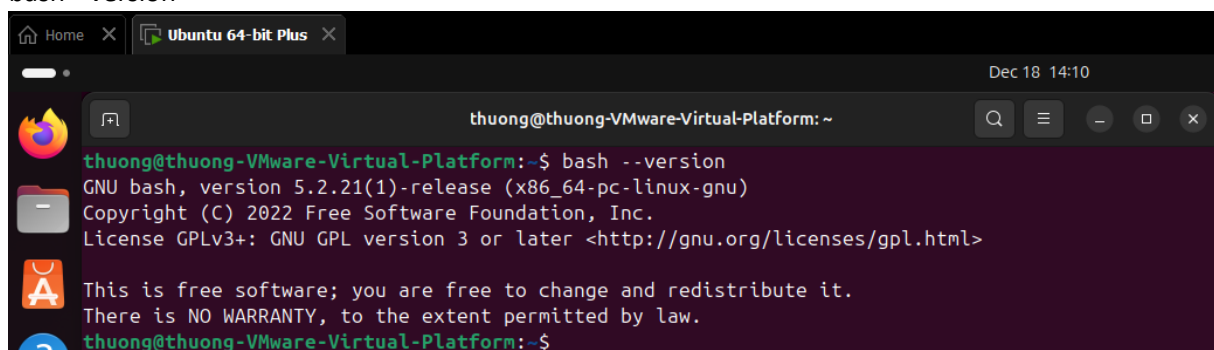
```
thuong@thuong-VMware-Virtual-Platform:~$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

python3 --version

A terminal window titled 'Ubuntu 64-bit Plus' with the user 'thuong@thuong-VMware-Virtual-Platform: ~'. The command 'python3 --version' has been executed, resulting in the output: 'Python 3.12.3'. A hint 'Try `python -h` for more information.' is also visible.

```
thuong@thuong-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
```

bash --version

A terminal window titled 'Ubuntu 64-bit Plus' with the user 'thuong@thuong-VMware-Virtual-Platform: ~'. The command 'bash --version' has been executed, resulting in the output: 'GNU bash, version 5.2.21(1)-release (x86\_64-pc-linux-gnu)', 'Copyright (C) 2022 Free Software Foundation, Inc.', and a license notice: 'License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>'. A further notice states: 'This is free software; you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.'.

```
thuong@thuong-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.
thuong@thuong-VMware-Virtual-Platform:~$
```

### Assignment 4.3: Compile

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

**Fibonacci.java and fib.c (Java and C are compiled languages. Their source code must be compiled before execution)**

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 and 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**

or **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**

Or

**bash fib.sh**

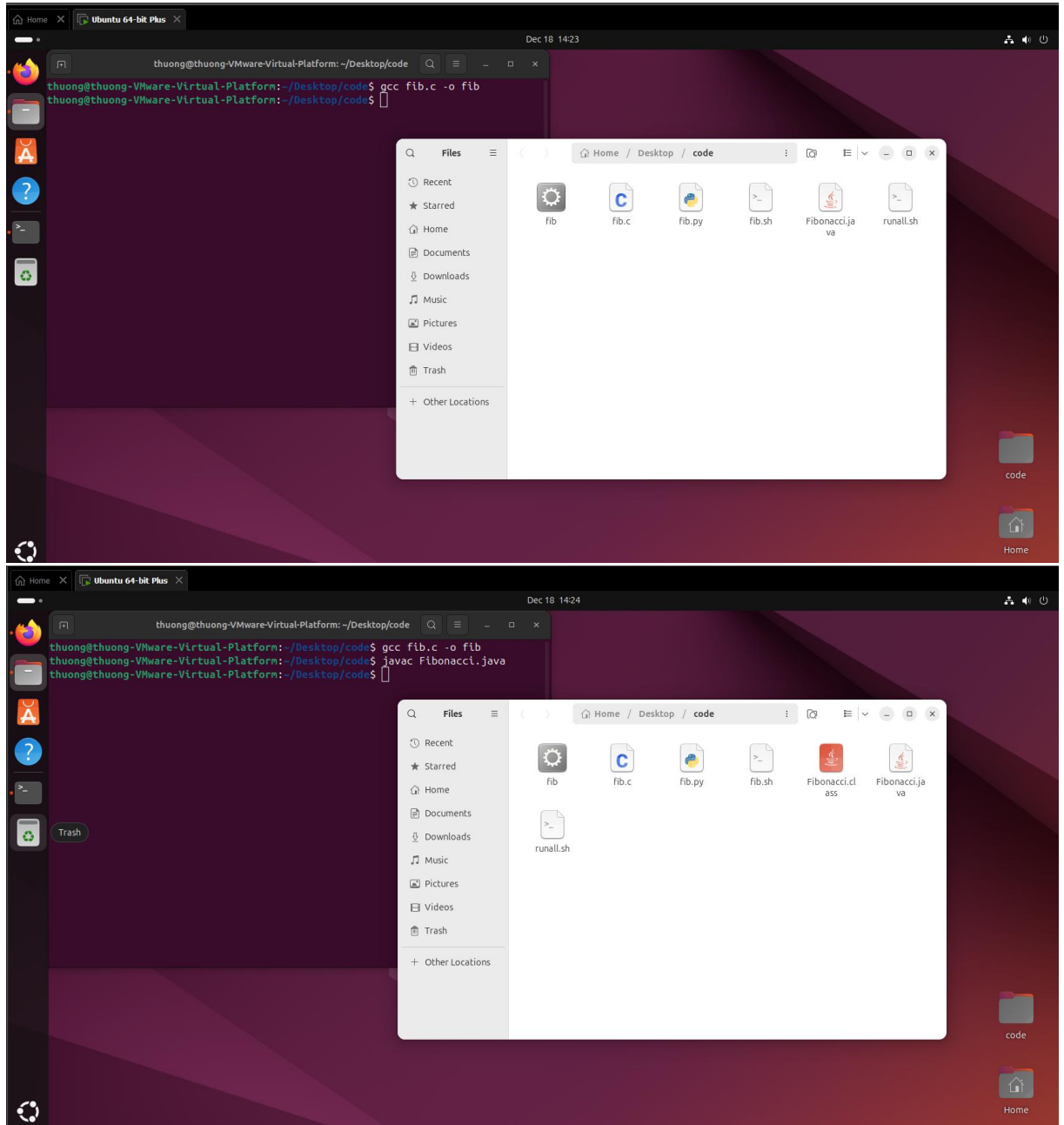
If I compile the above source code, will a new file be created? If so, which file?

Java -> Fibonacci.class

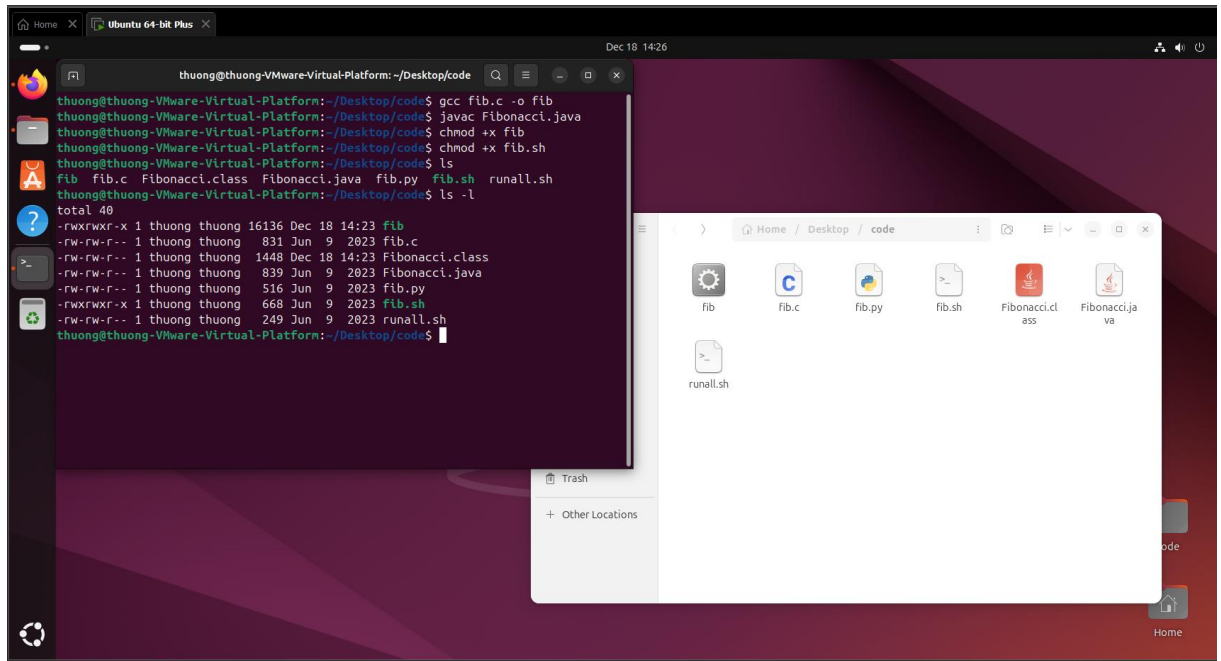
C -> fib (executable binary)

Take relevant screenshots of the following commands:

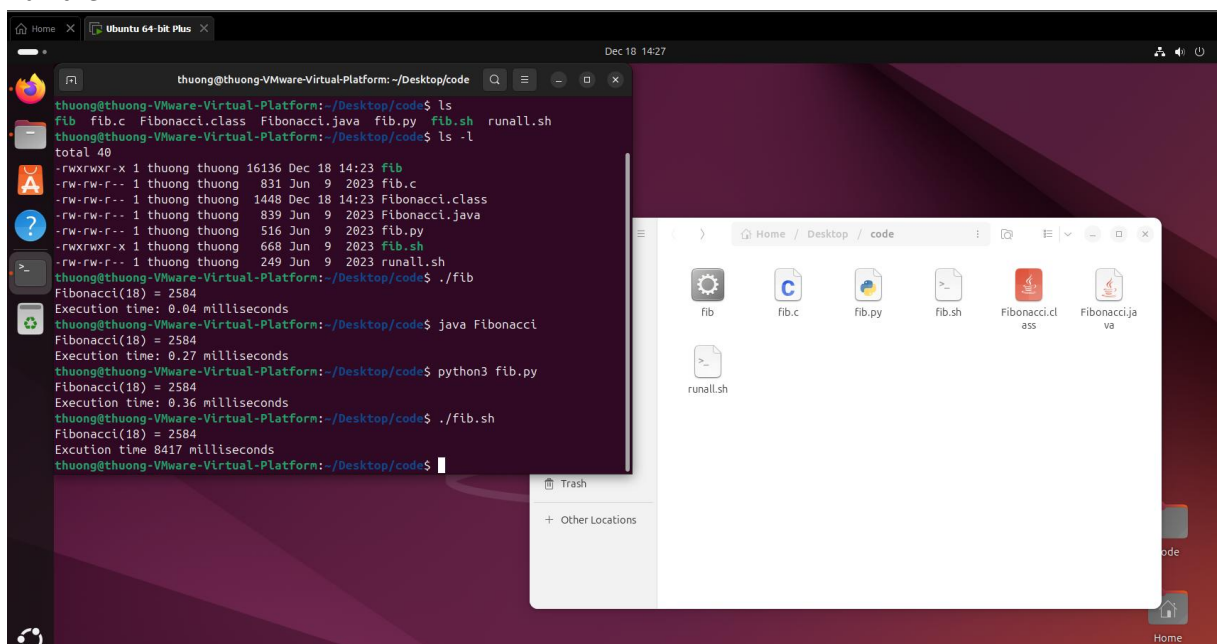
- Compile the source files where necessary



- Make them executable



- Run them



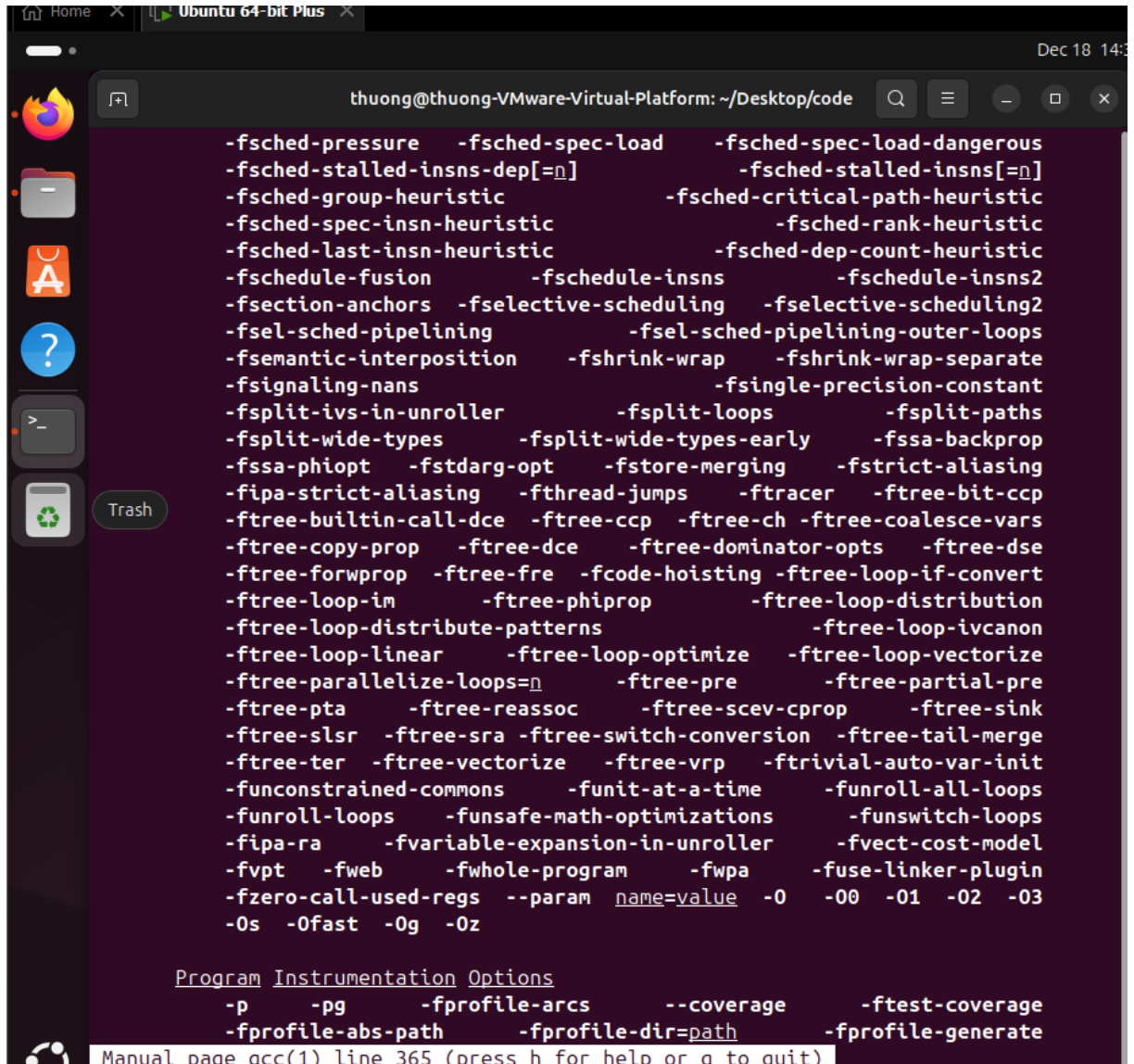
- Which (compiled) source code file performs the calculation the fastest?

**The C program (fib.c) performs the calculation the fastest because it is compiled into native machine code and executed directly by the processor**

## Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

- 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.



The screenshot shows a terminal window titled 'thuong@thuong-VMware-Virtual-Platform: ~/Desktop/code'. The terminal displays a list of GCC optimization options, organized in columns. The options include various flags for scheduling, code generation, and optimization levels. At the bottom, there is a section for 'Program Instrumentation Options' and a prompt to view the manual page for gcc(1).

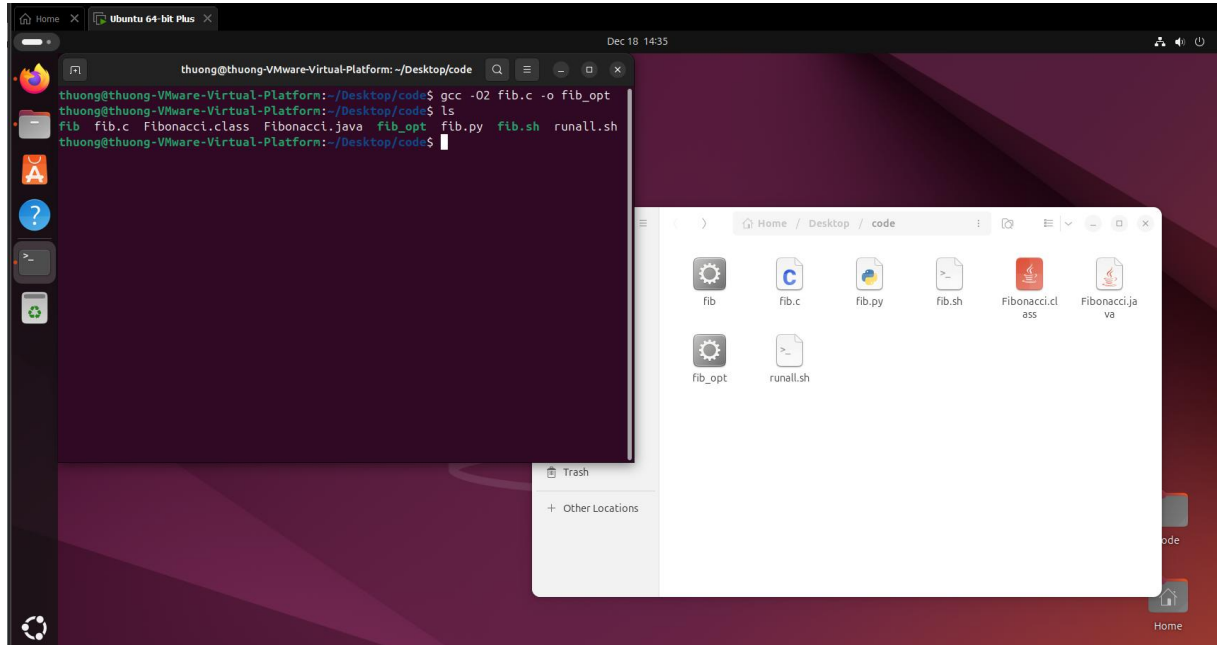
```
-fsched-pressure -fsched-spec-load -fsched-spec-load-dangerous
-fsched-stalled-insns-dep[=n] -fsched-stalled-insns[=n]
-fsched-group-heuristic -fsched-critical-path-heuristic
-fsched-spec-insn-heuristic -fsched-rank-heuristic
-fsched-last-insn-heuristic -fsched-dep-count-heuristic
-fschedule-fusion -fschedule-insns -fschedule-insns2
-fsection-anchors -fselective-scheduling -fselective-scheduling2
-fsel-sched-pipelining -fsel-sched-pipelining-outer-loops
-fsemantic-interposition -fshrink-wrap -fshrink-wrap-separate
-fsignaling-nans -fsingle-precision-constant
-fsplit-ivs-in-unroller -fsplit-loops -fsplit-paths
-fsplit-wide-types -fsplit-wide-types-early -fssa-backprop
-fssa-phiopt -fstdarg-opt -fstore-merging -fstrict-aliasing
-fipa-strict-aliasing -fthread-jumps -ftracer -ftree-bit-ccp
-ftree-builtin-call-dce -ftree-ccp -ftree-ch -ftree-coalesce-vars
-ftree-copy-prop -ftree-dce -ftree-dominator-opts -ftree-dse
-ftree-forwprop -ftree-fre -fcode-hoisting -ftree-loop-if-convert
-ftree-loop-in -ftree-phirop -ftree-loop-distribution
-ftree-loop-distribute-patterns -ftree-loop-ivcanon
-ftree-loop-linear -ftree-loop-optimize -ftree-loop-vectorize
-ftree-parallelize-loops=n -ftree-pre -ftree-partial-pre
-ftree-pta -ftree-reassoc -ftree-scev-cprop -ftree-sink
-ftree-slsr -ftree-sra -ftree-switch-conversion -ftree-tail-merge
-ftree-ter -ftree-vectorize -ftree-vrp -ftrivial-auto-var-init
-funconstrained-commons -funit-at-a-time -funroll-all-loops
-funroll-loops -funsafe-math-optimizations -funswitch-loops
-fipa-ra -fvariable-expansion-in-unroller -fvect-cost-model
-fvpt -fweb -fwhole-program -fwpa -fuse-linker-plugin
-fzero-call-used-regs --param name=value -O -O0 -O1 -O2 -O3
-Os -Ofast -Og -Oz

Program Instrumentation Options
-p -pg -fprofile-arcs --coverage -ftest-coverage
-fprofile-abs-path -fprofile-dir=path -fprofile-generate

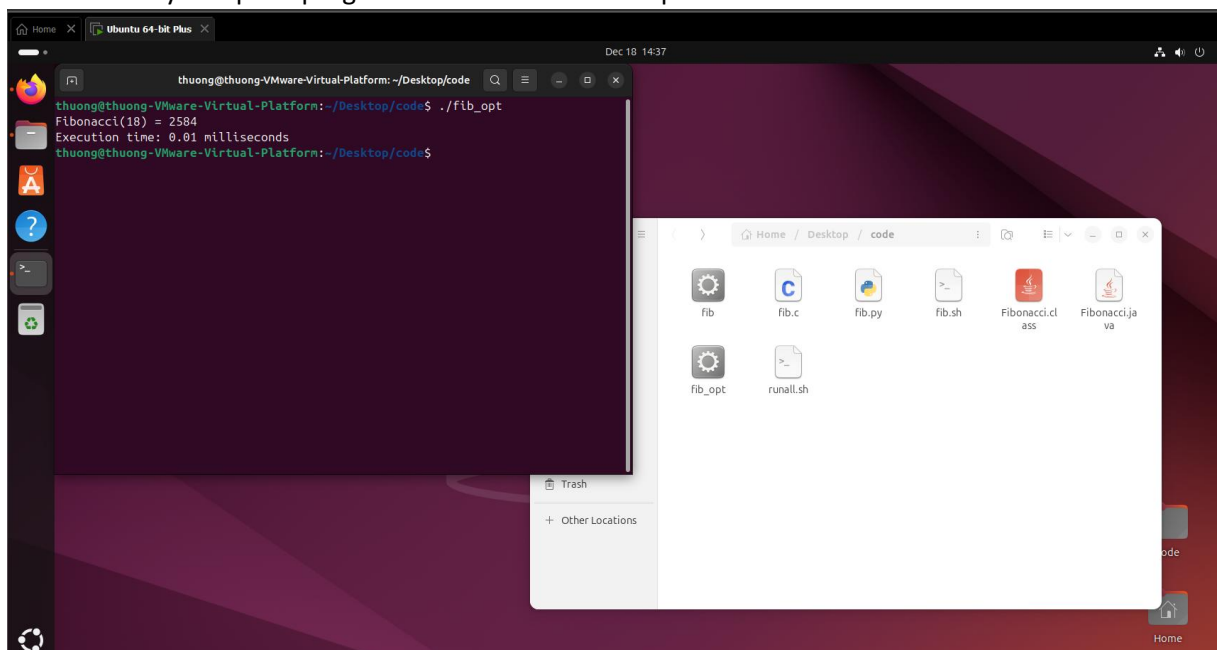
Manual page gcc(1) line 365 (press h for help or q to quit)
```

-O2 is an optimization parameter for the gcc compiler that enables a high level of optimization and improves execution performance without sacrificing correctness

- 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?

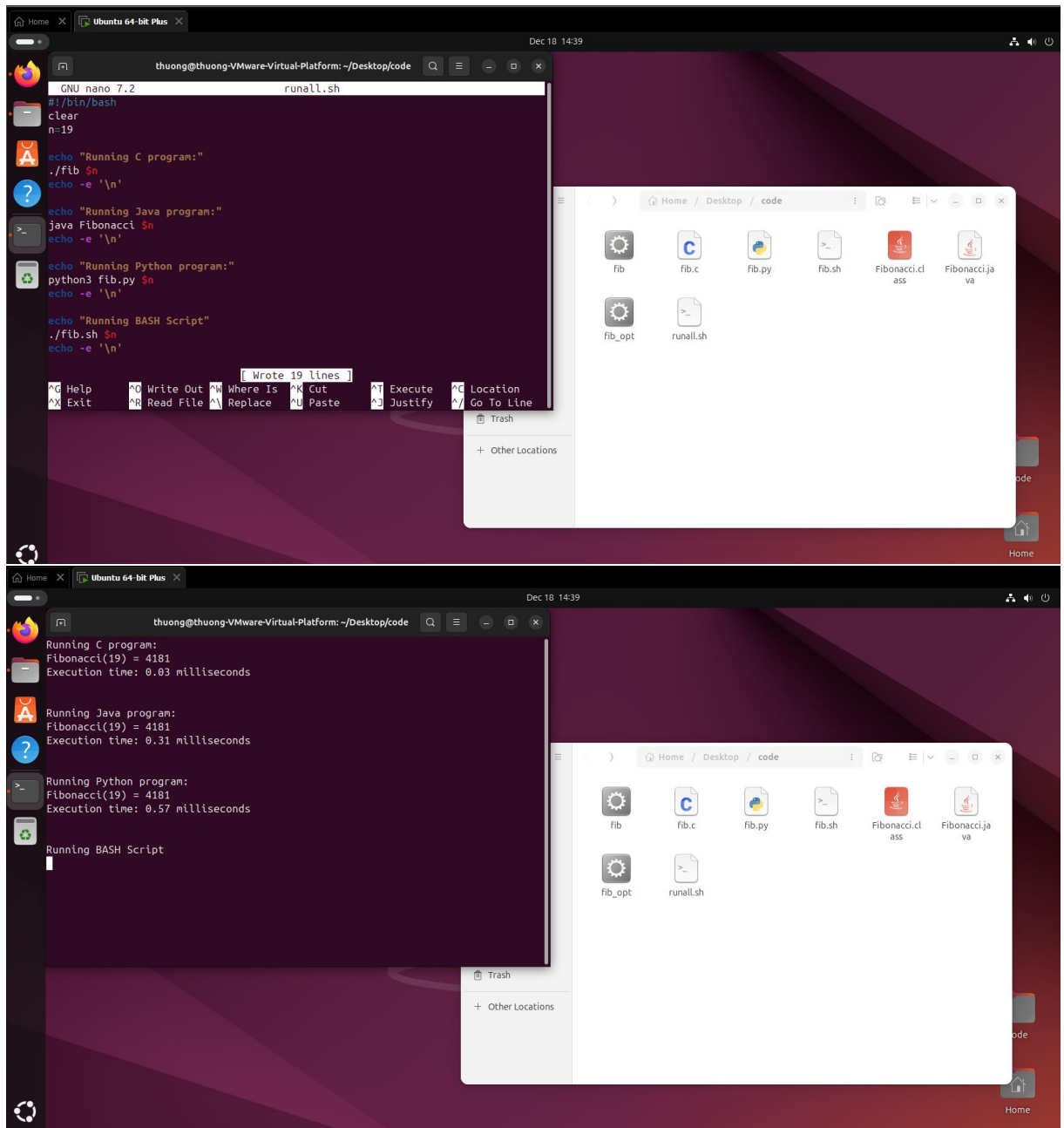


**Yes, the optimized C program performs the calculation faster.**

**After recompiling with the `-O2` optimization flag, the execution time decreased compared to the non-optimized version**

- 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
```

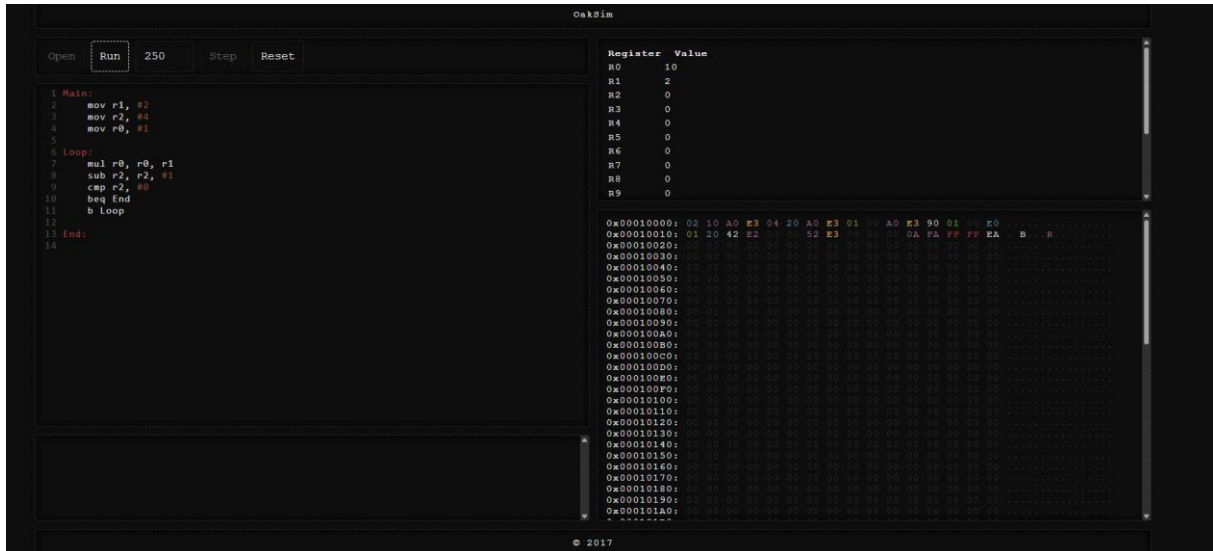
Loop:



End:

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

Screenshot of the completed code here.



Ready? Save this file and export it as a pdf file with the name: [week4.pdf](#)