

Week 4 – Software

Student number: 589932

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a debugger interface with the following details:

- Registers:** R0=0, R1=78, R2=1, R3=0, R4=0, R5=0, R6=0, R7=0, R8=0, R9=0.
- Memory Dump:** A large dump of memory starting at address 0x00010000, showing hex values for each byte.
- Assembly Code:** The code is as follows:

```
1 Main:
2     mov r2, #5
3     mov r1, #1
4
5 Loop:
6     mul r1, r1, r2
7     sub r2, r2, #1
8     cmp r2, #1
9     beq End
10    b Loop
11
12 End:
13
```

Assignment 4.2: Programming languages

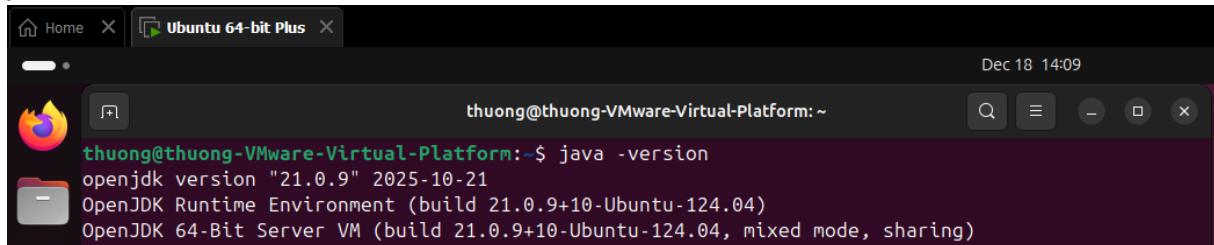
Take screenshots that the following commands work:

javac –version

The terminal window shows the following output:

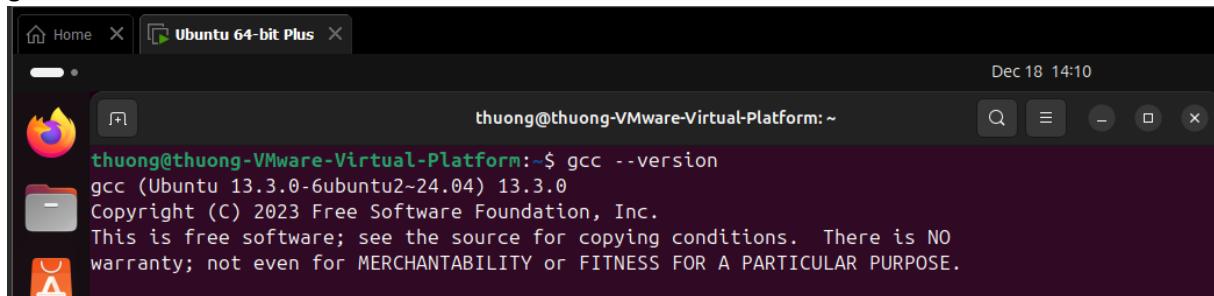
```
thuong@thuong-VMware-Virtual-Platform: ~
Usage: javac <options> <source files>
use --help for a list of possible options
thuong@thuong-VMware-Virtual-Platform: ~$ javac -version
javac 21.0.9
```

java -version



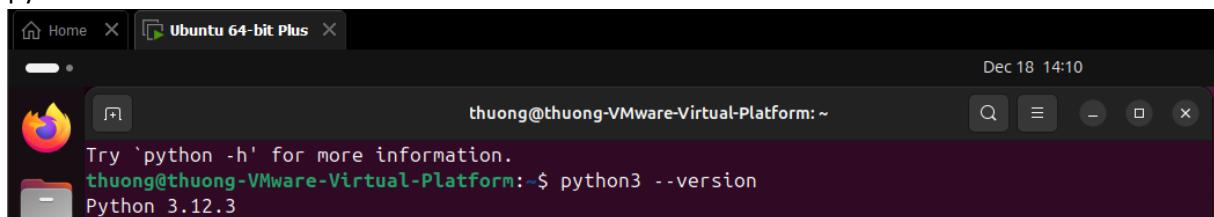
```
thuong@thuong-VMware-Virtual-Platform:~$ java -version
openjdk version "21.0.9" 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
```

gcc --version



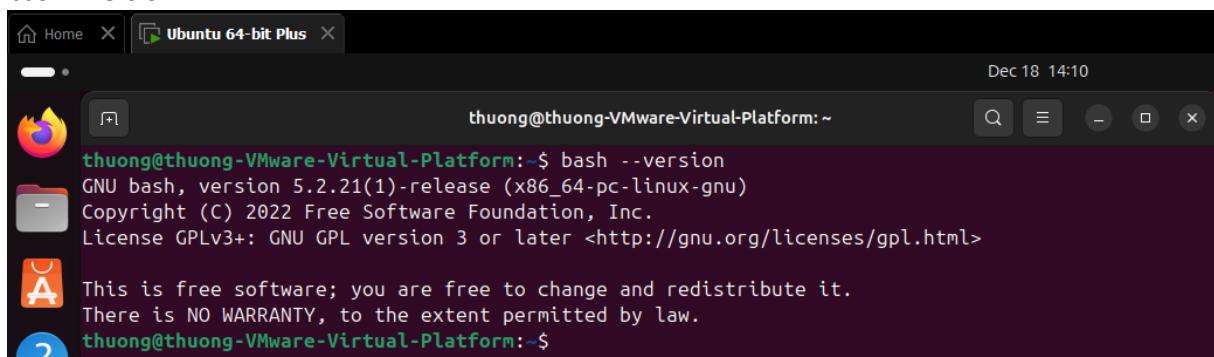
```
thuong@thuong-VMware-Virtual-Platform:~$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.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

python3 --version



```
Try `python -h` for more information.
thuong@thuong-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
```

bash --version



```
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

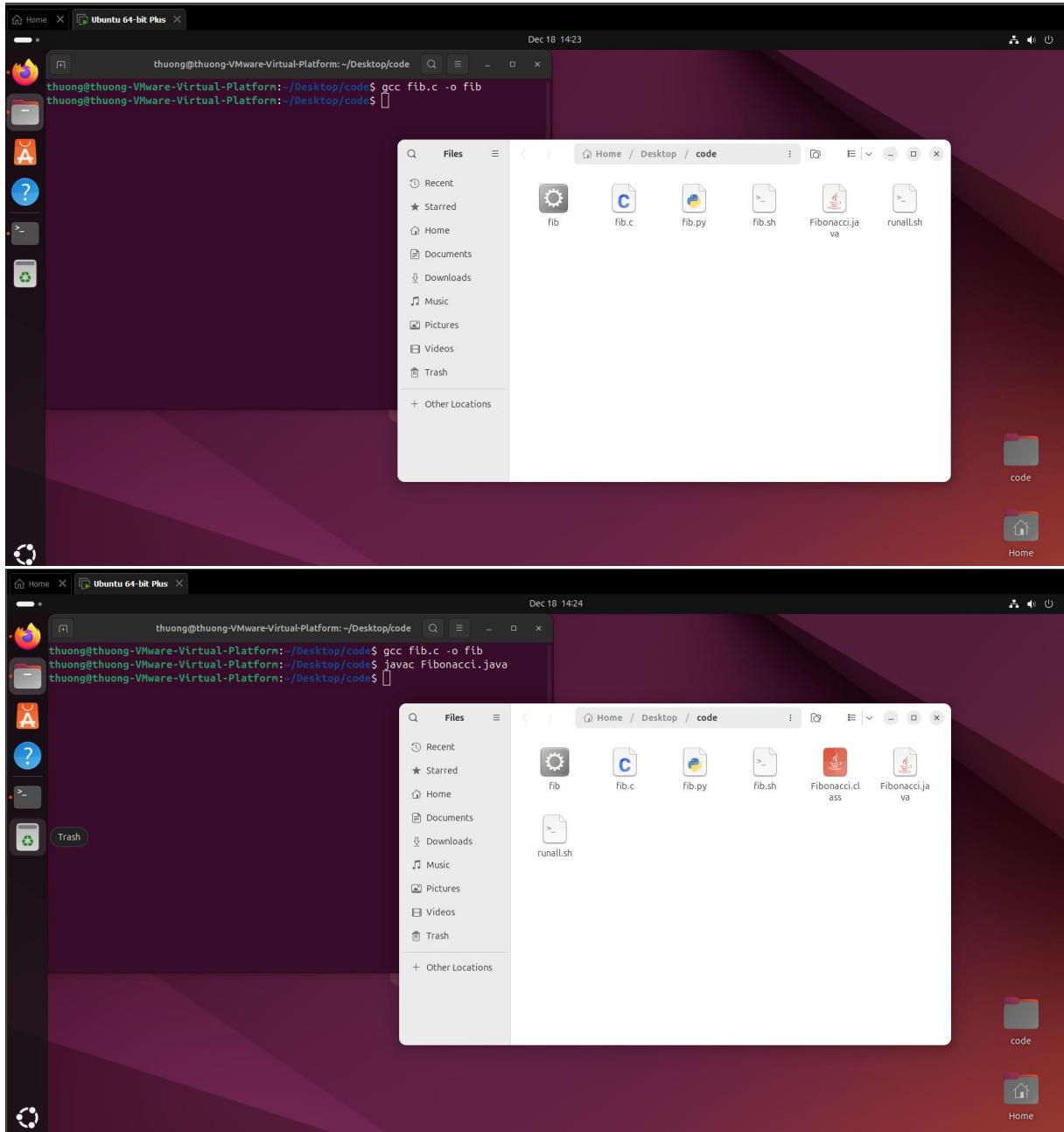
If I compile the above source code, will a new file be created? If so, which file?

Java -> Fibonacci.class

C -> fib (executable binary)

Take relevant screenshots of the following commands:

- Compile the source files where necessary



- Make them executable

The screenshot shows a Linux desktop environment with a terminal window and a file manager window.

Terminal Window:

```
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ gcc fib.c -o fib
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ javac Fibonacci.java
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ chmod +x fib
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ chmod +x fib.sh
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ ls -l
total 40
-rwxrwxr-x 1 thuong thuong 16136 Dec 18 14:23 fib
-rw-rw-r-- 1 thuong thuong 831 Jun  9 2023 fib.c
-rw-rw-r-- 1 thuong thuong 1448 Dec 18 14:23 Fibonacci.class
-rw-rw-r-- 1 thuong thuong 839 Jun  9 2023 Fibonacci.java
-rw-rw-r-- 1 thuong thuong 516 Jun  9 2023 fib.py
-rwxrwxr-x 1 thuong thuong 668 Jun  9 2023 fib.sh
-rw-rw-r-- 1 thuong thuong 249 Jun  9 2023 runall.sh
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$
```

File Manager Window:

The file manager shows the following files in the directory:

- fib
- fib.c
- fib.py
- fib.sh
- Fibonacci.class
- Fibonacci.java
- runall.sh

- Run them

The screenshot shows a Linux desktop environment with a terminal window and a file manager window.

Terminal Window:

```
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ ls -l
total 40
-rwxrwxr-x 1 thuong thuong 16136 Dec 18 14:23 fib
-rw-rw-r-- 1 thuong thuong 831 Jun  9 2023 fib.c
-rw-rw-r-- 1 thuong thuong 1448 Dec 18 14:23 Fibonacci.class
-rw-rw-r-- 1 thuong thuong 839 Jun  9 2023 Fibonacci.java
-rw-rw-r-- 1 thuong thuong 516 Jun  9 2023 fib.py
-rwxrwxr-x 1 thuong thuong 668 Jun  9 2023 fib.sh
-rw-rw-r-- 1 thuong thuong 249 Jun  9 2023 runall.sh
Fibonacci(18) = 2584
Execution time: 0.04 milliseconds
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.36 milliseconds
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 8417 milliseconds
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code$
```

File Manager Window:

The file manager shows the following files in the directory:

- fib
- fib.c
- fib.py
- fib.sh
- Fibonacci.class
- Fibonacci.java
- runall.sh

- 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:

- 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.

```
thuong@thuong-VMware-Virtual-Platform:~/Desktop/code
```

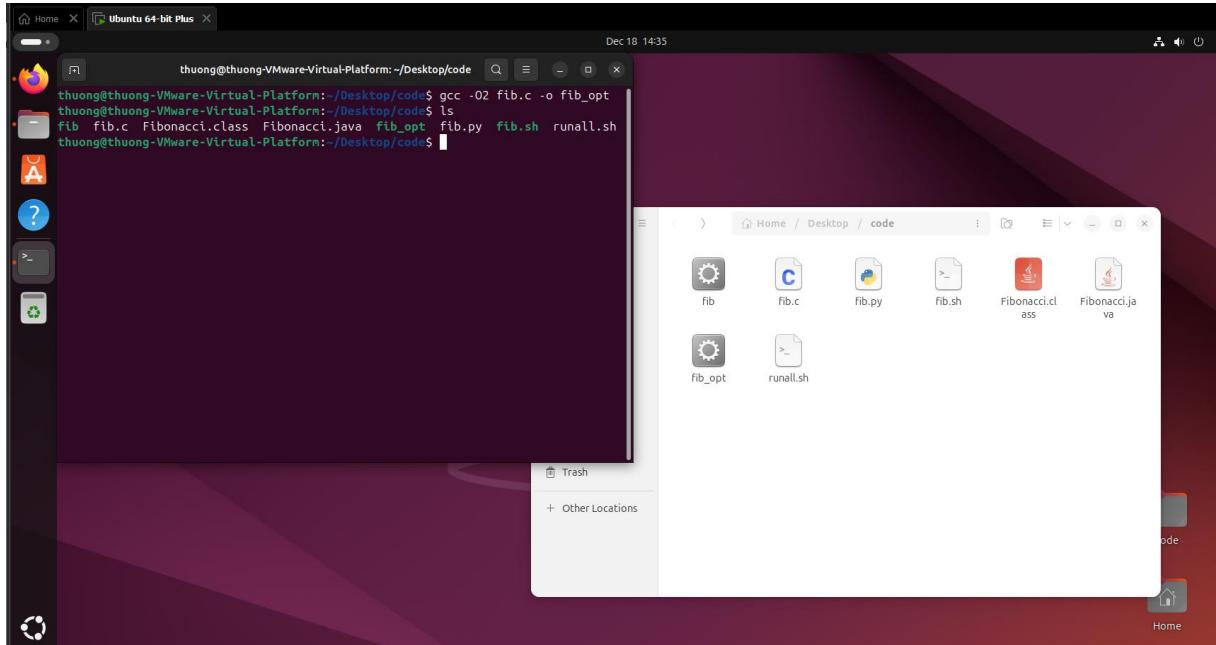
-fsched-pressure -fsched-spec-load -fsched-spec-load-dangerous
-fsched-stalled-insns-dep[=n] -fsched-stalled-insns[=n]
-fsched-group-heuristic -fsched-critical-path-heuristic
-fsched-spec-insn-heuristic -fsched-rank-heuristic
-fsched-lastInsnHeuristic -fsched-dep-count-heuristic
-fschedule-fusion -fschedule-insns -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-im -ftree-phiprop -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-ptx -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-reg -param name=value -O0 -O0 -O1 -O2 -O3
-Os -Ofast -Og -Oz

Program Instrumentation Options

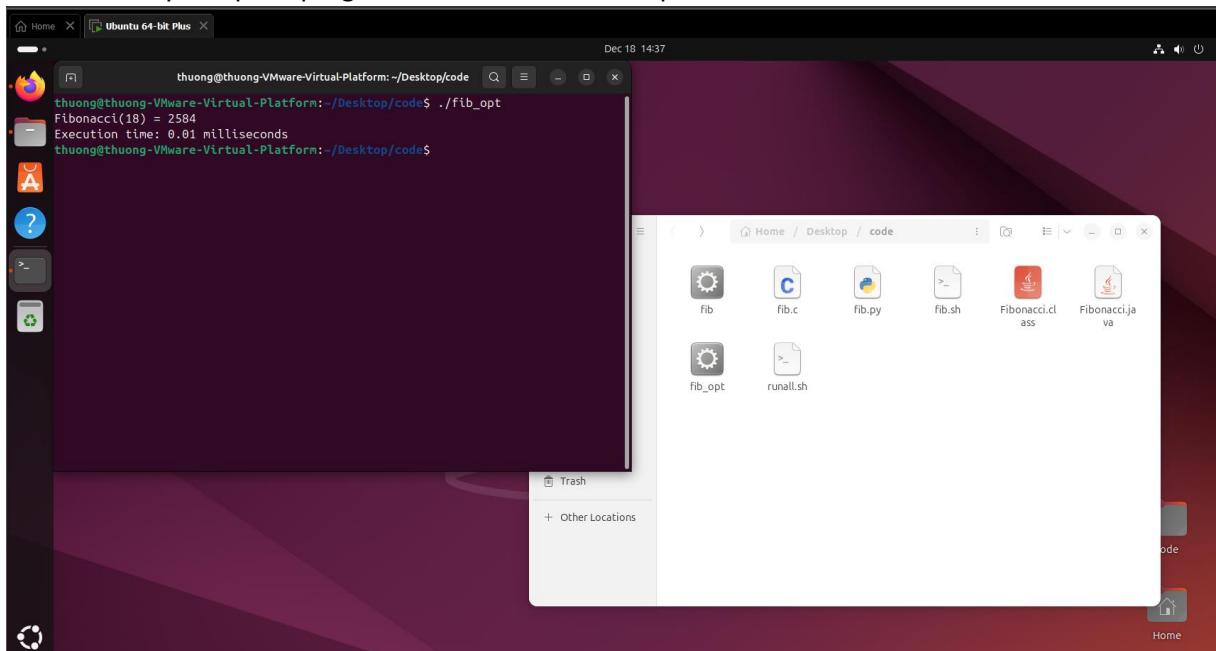
-p -pg -fprofile-arcs --coverage -ftest-coverage
-fprofile-abs-path -fprofile-dir=path -fprofile-generate

-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



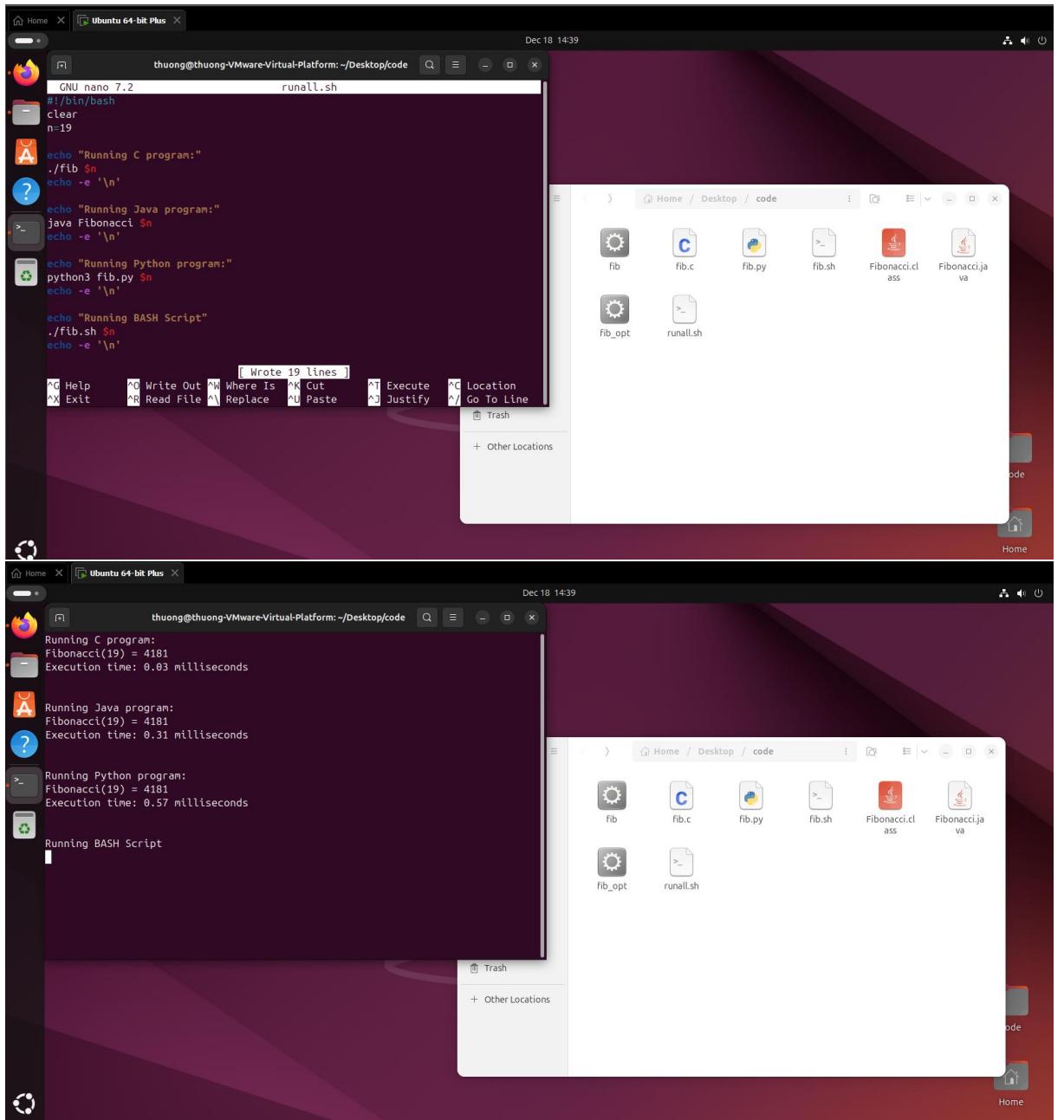
- c) Run the newly compiled program. Is it true that it now performs the calculation faster?



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.

The screenshot shows the OakSim assembly debugger interface. At the top, there are buttons for 'Open', 'Run' (which is currently selected), '250', 'Step', and 'Reset'. Below the buttons is the assembly code:

```
1 Main:
2     mov r1, #2
3     mov r2, #4
4     mov r0, #1
5
6 Loop:
7     mul r0, r0, r1
8     sub r2, r2, #1
9     cmp r2, #0
10    bne End
11    b Loop
12
13 End:
14
```

On the right side, there is a register table titled 'Register Value' and a memory dump table. The register table shows:

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

The memory dump table shows memory starting at address 0x00010000. The first few lines of memory are:

Address	Value
0x00010000	02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 90 01 00 E0 . . .
0x00010010	01 20 42 E2 00 00 52 E3 00 00 0A FA FF FF EA . . .
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 . . .

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