Template Week 4 – Software

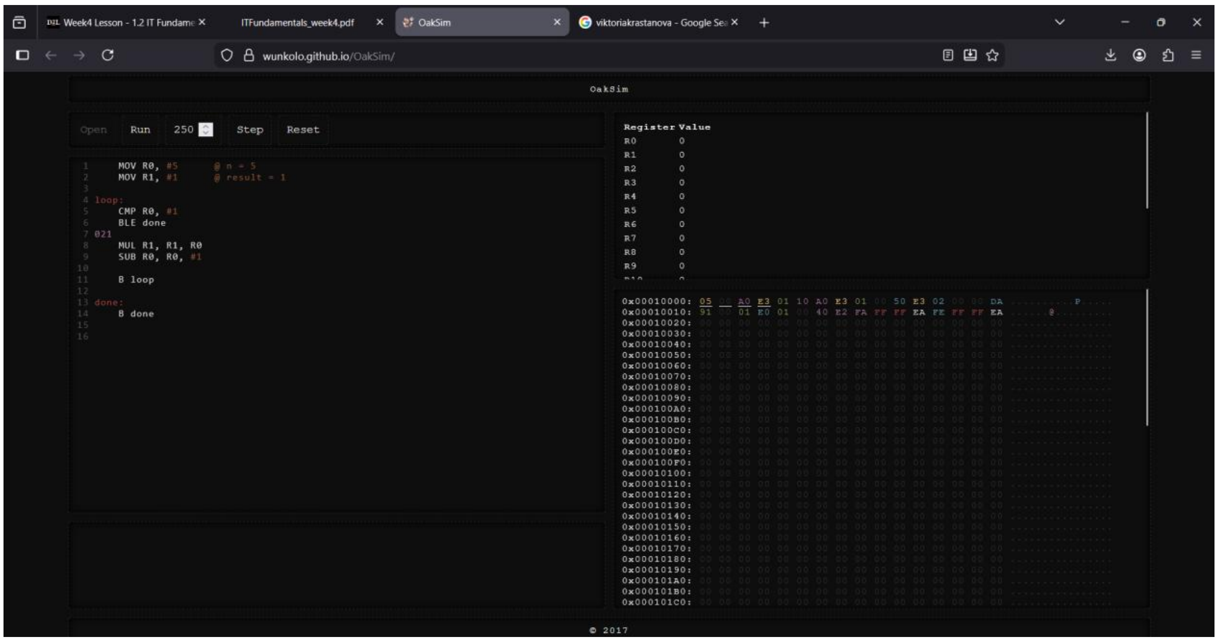
Student number:

582122

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

My name is in a new tab



```
    MOV R0, #5       @ n = 5
    MOV R1, #1       @ result = 1

loop:
    CMP R0, #1
    BLE done
021
    MUL R1, R1, R0
    SUB R0, R0, #1

    B loop

done:
    B done
```

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

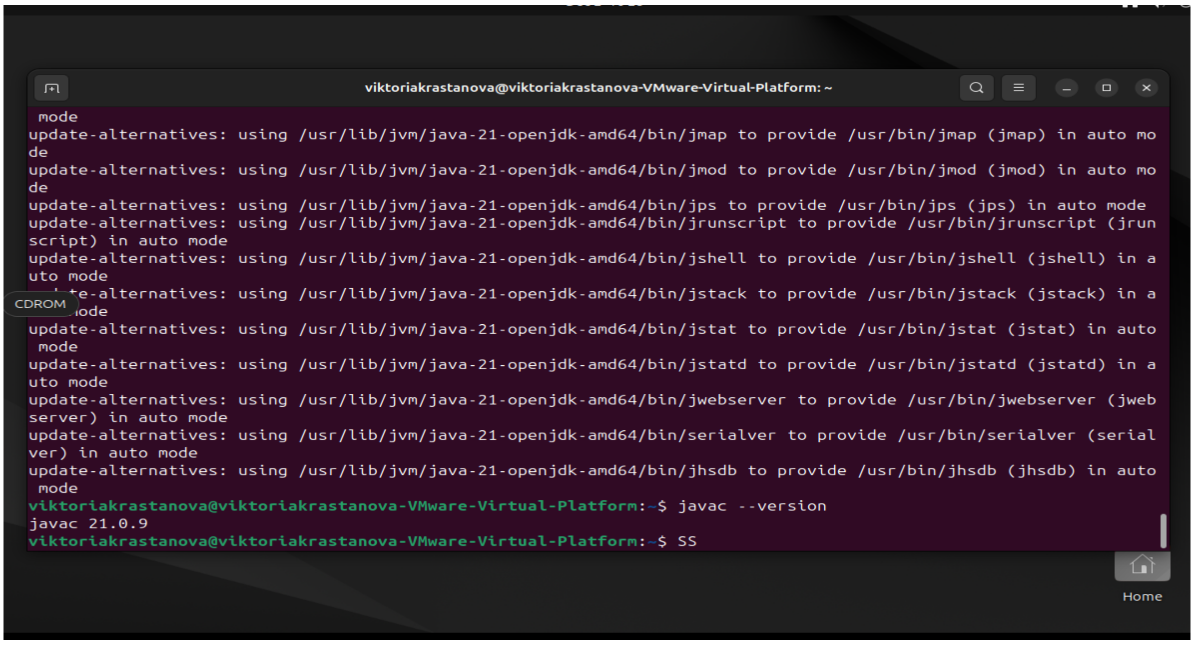
```
0x00010000: 05 00 A0 E3 01 10 A0 E3 01 00 50 E3 02 00 00 DA  ..........P.....
0x00010010: 91 00 01 E0 01 00 40 E2 FA FF FF EA FE FF FF EA  ......@.........
0x00010020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
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  ................
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  ................
```

Week4 Lesson - 1.2 IT Fundame
ITFundamentals_week4.pdf
OakSim
viktoriakrastanova - Google Sea
wunkolo.github.io/OakSim/
OakSim
Open
Run
250
Step
Reset
© 2017

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

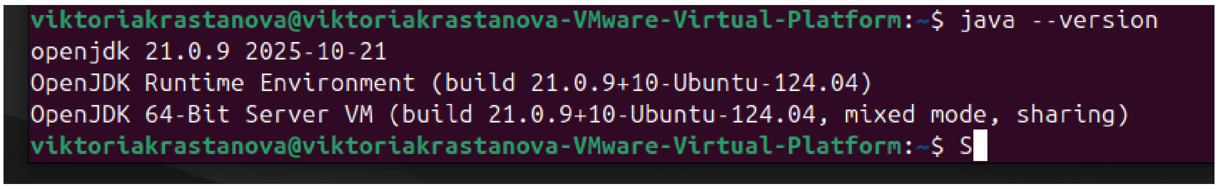
javac –version



```
 mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jmap to provide /usr/bin/jmap (jmap) in auto mo
de
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jmod to provide /usr/bin/jmod (jmod) in auto mo
de
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jps to provide /usr/bin/jps (jps) in auto mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jrunscript to provide /usr/bin/jrunscript (jrun
script) in auto mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jshell to provide /usr/bin/jshell (jshell) in a
uto mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jstack to provide /usr/bin/jstack (jstack) in a
uto mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jstat to provide /usr/bin/jstat (jstat) in auto
 mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jstatd to provide /usr/bin/jstatd (jstatd) in a
uto mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jwebserver to provide /usr/bin/jwebserver (jweb
server) in auto mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/serialver to provide /usr/bin/serialver (serial
ver) in auto mode
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jhsdb to provide /usr/bin/jhsdb (jhsdb) in auto
 mode
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ SS
```

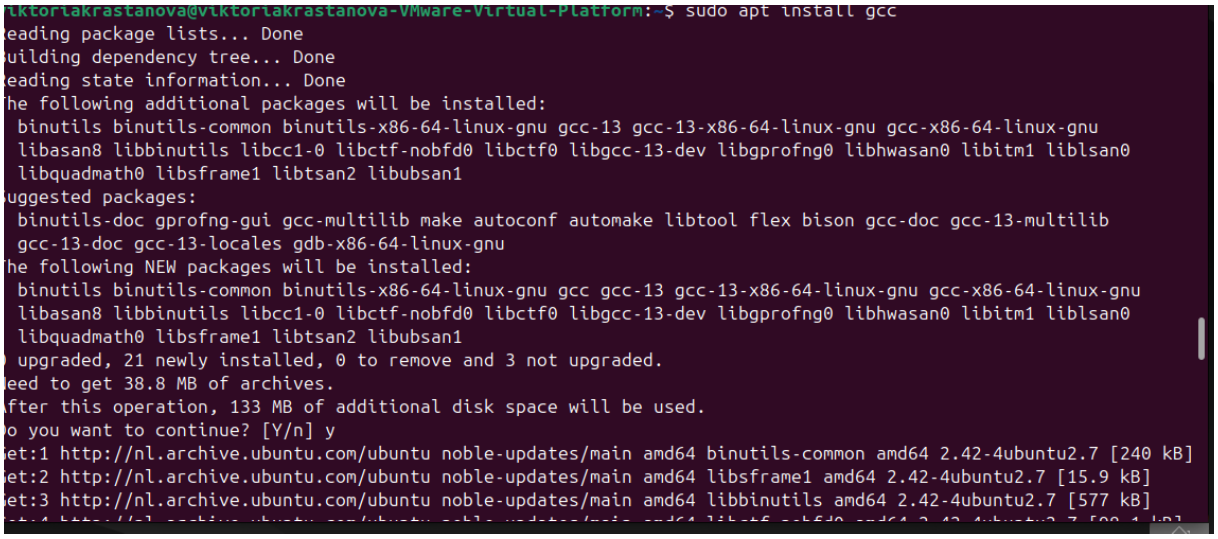
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform: ~
CDROM
Home

java –version



```
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ java --version
openjdk 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)
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ S
```

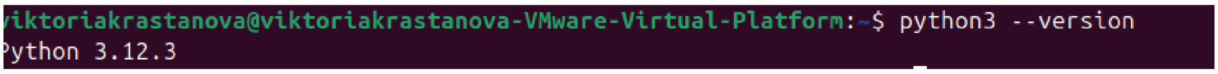
gcc –version



```
iktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ sudo apt install gcc
eading package lists... Done
uilding dependency tree... Done
eading state information... Done
he following additional packages will be installed:
 binutils binutils-common binutils-x86-64-linux-gnu gcc-13 gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu
 libasan8 libbinutils libcc1-0 libctf-nobfd0 libctf0 libgcc-13-dev libgprofng0 libhwasan0 libitm1 liblsan0
 libquadmath0 libsframe1 libtsan2 libubsan1
uggested packages:
 binutils-doc gprofng-gui gcc-multilib make autoconf automake libtool flex bison gcc-doc gcc-13-multilib
 gcc-13-doc gcc-13-locales gdb-x86-64-linux-gnu
he following NEW packages will be installed:
 binutils binutils-common binutils-x86-64-linux-gnu gcc gcc-13 gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu
 libasan8 libbinutils libcc1-0 libctf-nobfd0 libctf0 libgcc-13-dev libgprofng0 libhwasan0 libitm1 liblsan0
 libquadmath0 libsframe1 libtsan2 libubsan1
 upgraded, 21 newly installed, 0 to remove and 3 not upgraded.
eed to get 38.8 MB of archives.
fter this operation, 133 MB of additional disk space will be used.
o you want to continue? [Y/n] y
et:1 http://nl.archive.ubuntu.com/ubuntu noble-updates/main amd64 binutils-common amd64 2.42-4ubuntu2.7 [240 kB]
et:2 http://nl.archive.ubuntu.com/ubuntu noble-updates/main amd64 libsframe1 amd64 2.42-4ubuntu2.7 [15.9 kB]
et:3 http://nl.archive.ubuntu.com/ubuntu noble-updates/main amd64 libbinutils amd64 2.42-4ubuntu2.7 [577 kB]
```

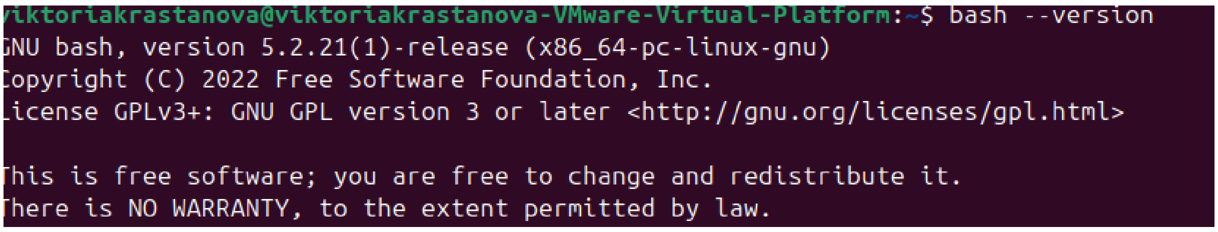
python3 –version

installed by default so =>



```
iktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ python3 --version
ython 3.12.3
```

bash –version



```
iktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ bash --version
NU bash, version 5.2.21(1)-release (x86_64-pc-linux-gnu)
opyright (C) 2022 Free Software Foundation, Inc.
icense GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

his is free software; you are free to change and redistribute it.
here is NO WARRANTY, to the extent permitted by law.
```

Assignment 4.3: Compile

Which of the files need to be compiled before you can run them?
The files that need to be compiled are fib.c and Fibonacci.java. The C file fib.c must be compiled into machine code before I can run it, and the Java file is normally compiled into bytecode, although newer versions of Java can run a single file directly without a separate compile step. The Python file fib.py and the bash script fib.sh are interpreted and do not need compilation.

Which source code files are compiled into machine code and are directly executable by a processor?
Only the C file fib.c is compiled into machine code that the processor can execute directly.

Which source code files are compiled to bytecode?
The Java file Fibonacci.java is compiled into bytecode that runs on the Virtual Machine. Python files are also compiled into bytecode behind the scenes, but they are executed by the Python interpreter rather than directly by the processor.

Which source code files are interpreted by an interpreter?
The Python file fib.py is interpreted by the Python interpreter, and the bash script fib.sh is interpreted by the shell. The Java bytecode is also executed by the JVM, which acts as an interpreter or just-in-time compiler.

Which one is expected to perform the calculation the fastest?
The C program fib.c is expected to perform the calculation the fastest because it runs directly as native machine code. Java is usually slower than C because of JVM overhead. Python and bash are generally the slowest because they are interpreted.

How do I run a Java program?
You first compile it using javac Fibonacci.java, which produces a file called Fibonacci.class, and then run it with java Fibonacci.

How do I run a Python program?
You use python3 fib.py. If the file has a shebang line and is made executable with chmod +x fib.py, you can also run it with ./fib.py.

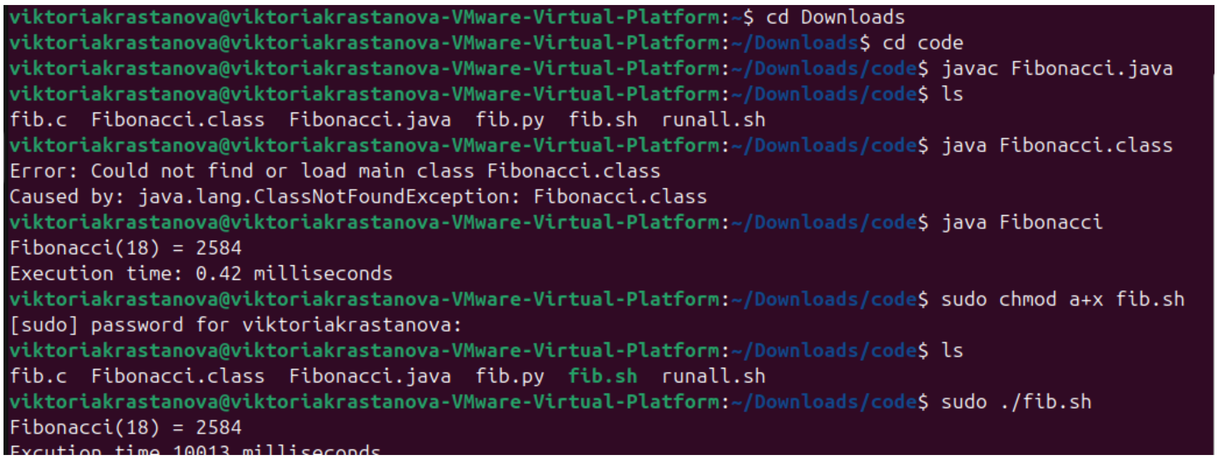
How do I run a C program?
You compile it with gcc -o fib fib.c, which creates an executable called fib, and then run it with ./fib.

How do I run a bash script?
You can make it executable with chmod +x fib.sh and then run it with ./fib.sh or simply run it with bash fib.sh.

If I compile the above source code, will a new file be created? If so, which file?
Compiling the C program creates a new executable file containing machine code. Compiling the Java program creates a .class file containing bytecode. Python usually doesn’t create a new file unless explicitly compiled or cached by the interpreter, and bash scripts do not produce a new file at all.

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?

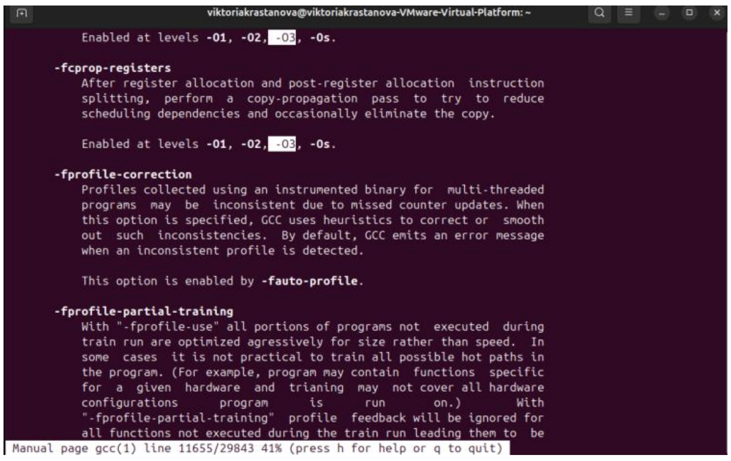


```
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~$ cd Downloads
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads$ cd code
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci.class
Error: Could not find or load main class Fibonacci.class
Caused by: java.lang.ClassNotFoundException: Fibonacci.class
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.42 milliseconds
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x fib.sh
[sudo] password for viktoriakrastanova:
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 10013 milliseconds
```

Assignment 4.4: Optimize

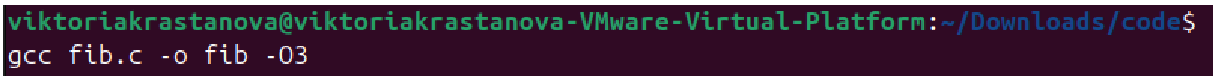
Take relevant screenshots of the following commands:

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.



```
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform: ~

           Enabled at levels -O1, -O2, -O3, -Os.

       -fcprop-registers
           After register allocation and post-register allocation  instruction
           splitting,  perform  a  copy-propagation  pass  to  try  to  reduce
           scheduling dependencies and occasionally eliminate the copy.

           Enabled at levels -O1, -O2, -O3, -Os.

       -fprofile-correction
           Profiles collected using an instrumented binary for  multi-threaded
           programs  may  be  inconsistent due to missed counter updates. When
           this option is specified, GCC uses heuristics to correct or  smooth
           out  such  inconsistencies.  By default, GCC emits an error message
           when an inconsistent profile is detected.

           This option is enabled by -fauto-profile.

       -fprofile-partial-training
           With "-fprofile-use" all portions of programs not  executed  during
           train run are optimized agressively for size rather than speed.  In
           some  cases  it is not practical to train all possible hot paths in
           the program. (For example, program may contain  functions  specific
           for  a  given  hardware  and  trianing  may  not cover all hardware
           configurations      program      is      run       on.)       With
           "-fprofile-partial-training"  profile  feedback will be ignored for
           all functions not executed during the train run leading them to  be
 Manual page gcc(1) line 11655/29843 41% (press h for help or q to quit)
```

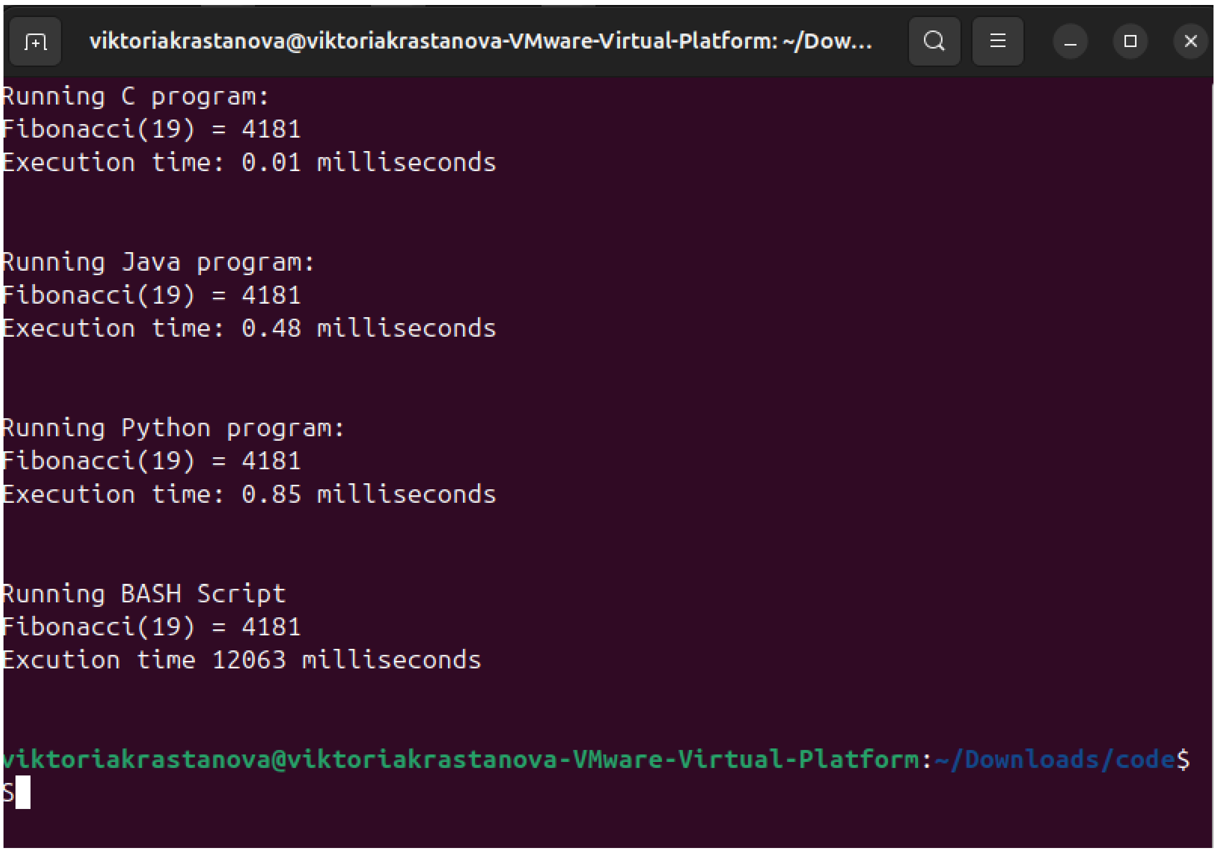
b) Compile **fib.c** again with the optimization parameters



```
viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform:~/Downloads/code$
gcc fib.c -o fib -O3
```

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

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.



viktoriakrastanova@viktoriakrastanova-VMware-Virtual-Platform: ~/Dow...

```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.48 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.85 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 12063 milliseconds


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

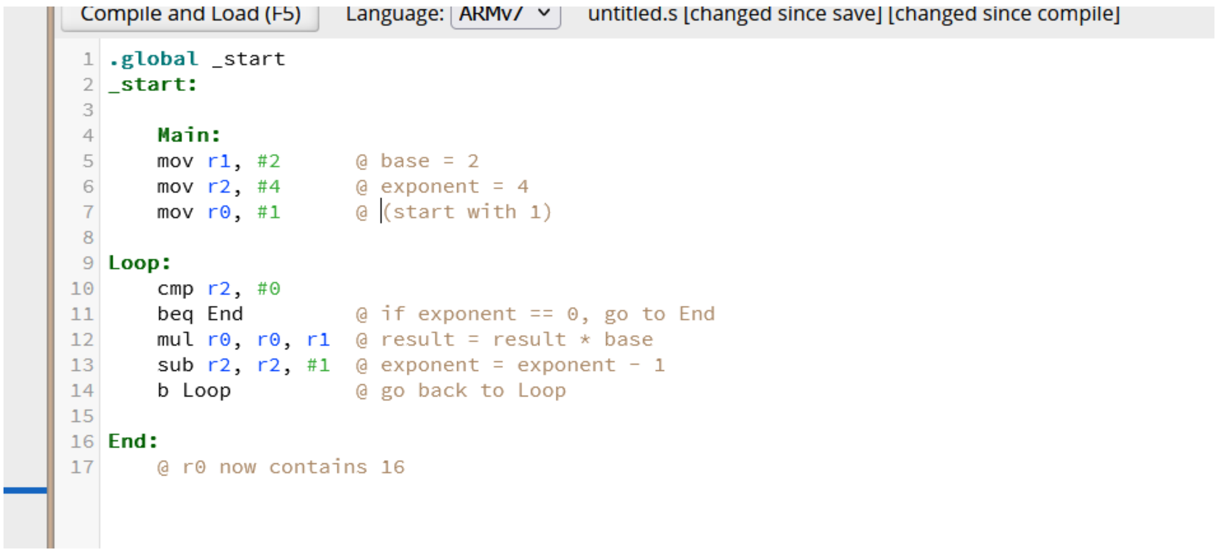
Assignment 4.5: More ARM Assembly

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

Loop:

End:
```

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

Screenshot of the completed code here.



```
.global _start
_start:

    Main:
    mov r1, #2        @ base = 2
    mov r2, #4        @ exponent = 4
    mov r0, #1        @ (start with 1)

Loop:
    cmp r2, #0
    beq End           @ if exponent == 0, go to End
    mul r0, r0, r1    @ result = result * base
    sub r2, r2, #1    @ exponent = exponent - 1
    b Loop            @ go back to Loop

End:
    @ r0 now contains 16
```

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