

Week 4 – Software

Student number: 578438

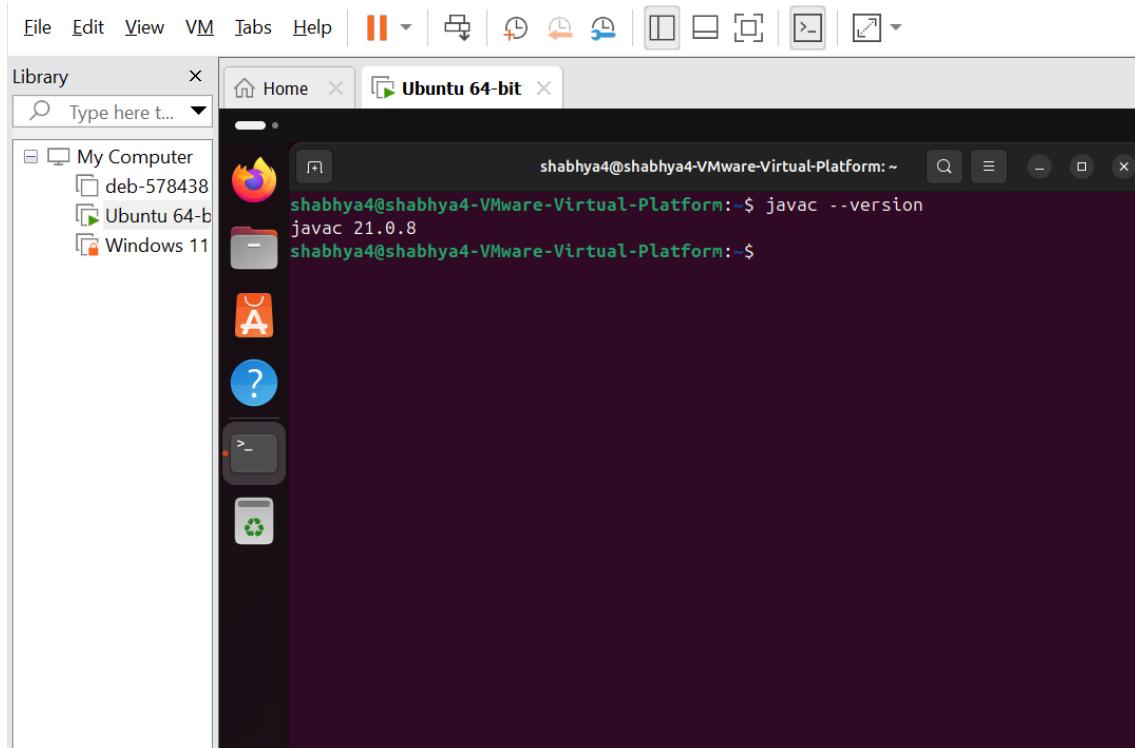
Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

Assignment 4.2: Programming languages

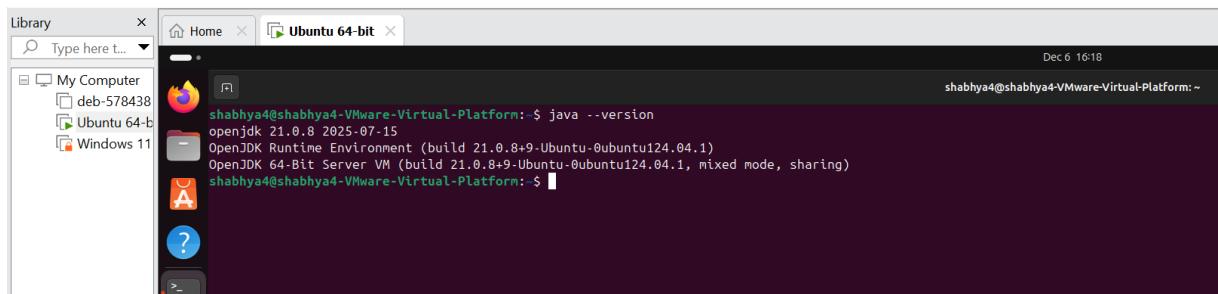
Take screenshots that the following commands work:

javac --version



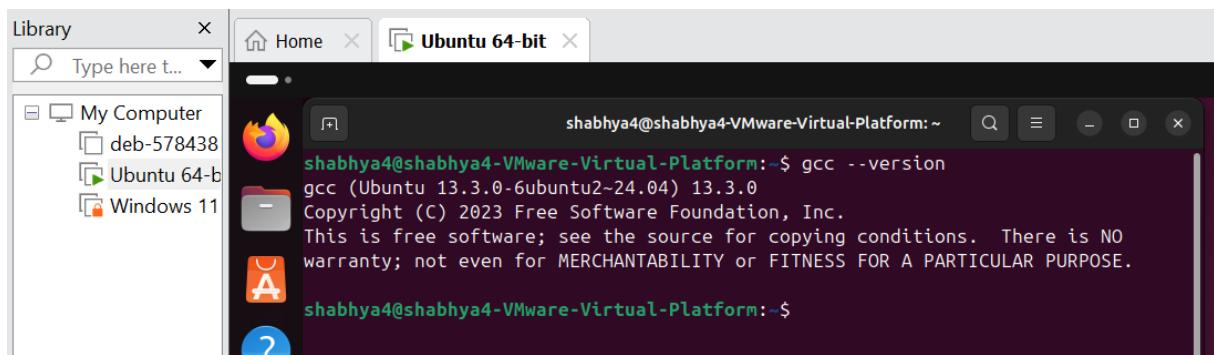
A screenshot of a Linux desktop environment. On the left is a file explorer window titled "Library" showing "My Computer" with icons for "deb-578438", "Ubuntu 64-bit", and "Windows 11". To the right is a terminal window titled "Ubuntu 64-bit" with the command "javac --version" run, displaying the output "javac 21.0.8".

java --version



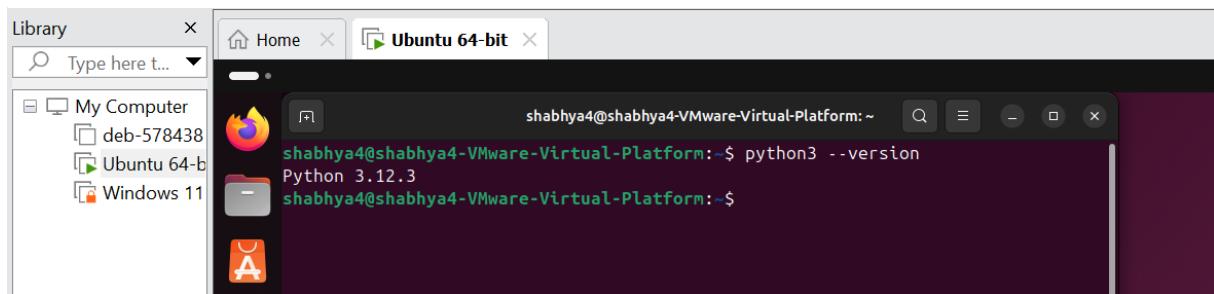
A screenshot of a Linux desktop environment. On the left is a file explorer window titled "Library" showing "My Computer" with icons for "deb-578438", "Ubuntu 64-bit", and "Windows 11". To the right is a terminal window titled "Ubuntu 64-bit" with the command "java --version" run, displaying the output "openjdk 21.0.8 2025-07-15" and "OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu124.04.1)" and "OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)".

gcc --version



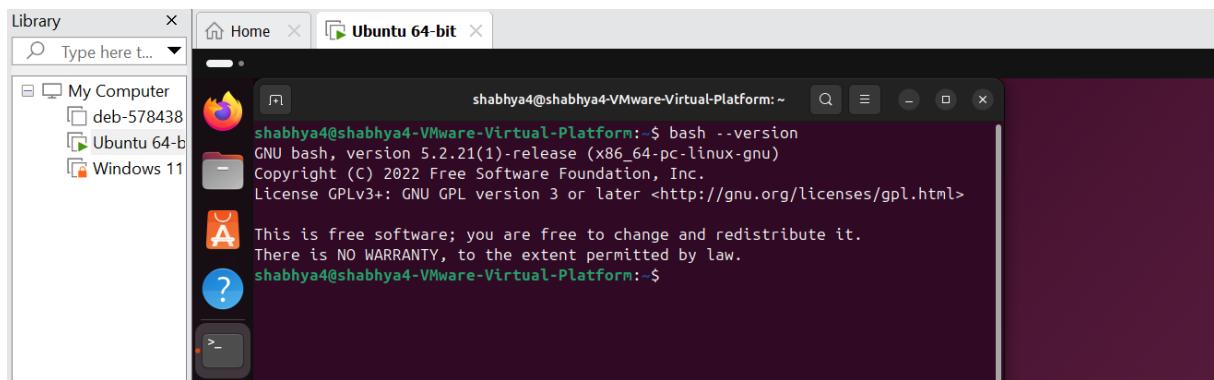
```
shabhy4@shabhy4-VMware-Virtual-Platform:~$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

python3 --version



```
shabhy4@shabhy4-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
shabhy4@shabhy4-VMware-Virtual-Platform:~$
```

bash --version



```
shabhy4@shabhy4-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.
shabhy4@shabhy4-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

Which source code files are compiled into machine code and then directly executable by a processor?

- The C program fib.c, after compilation.

Which source code files are compiled to byte code?

- The Java program Fibonacci.java → Fibonacci.class.

Which source code files are interpreted by an interpreter?

- The Python program fib.py and the Bash script “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 compiled C program.

How do I run a Java program?

- Compile: javac Fibonacci.java
- Run:

How do I run a Python program?

- Python3 fib.py

How do I run a C program?

- Compile: gcc fib.c -o fib_c
- Run: ./fib_c

How do I run a Bash script?

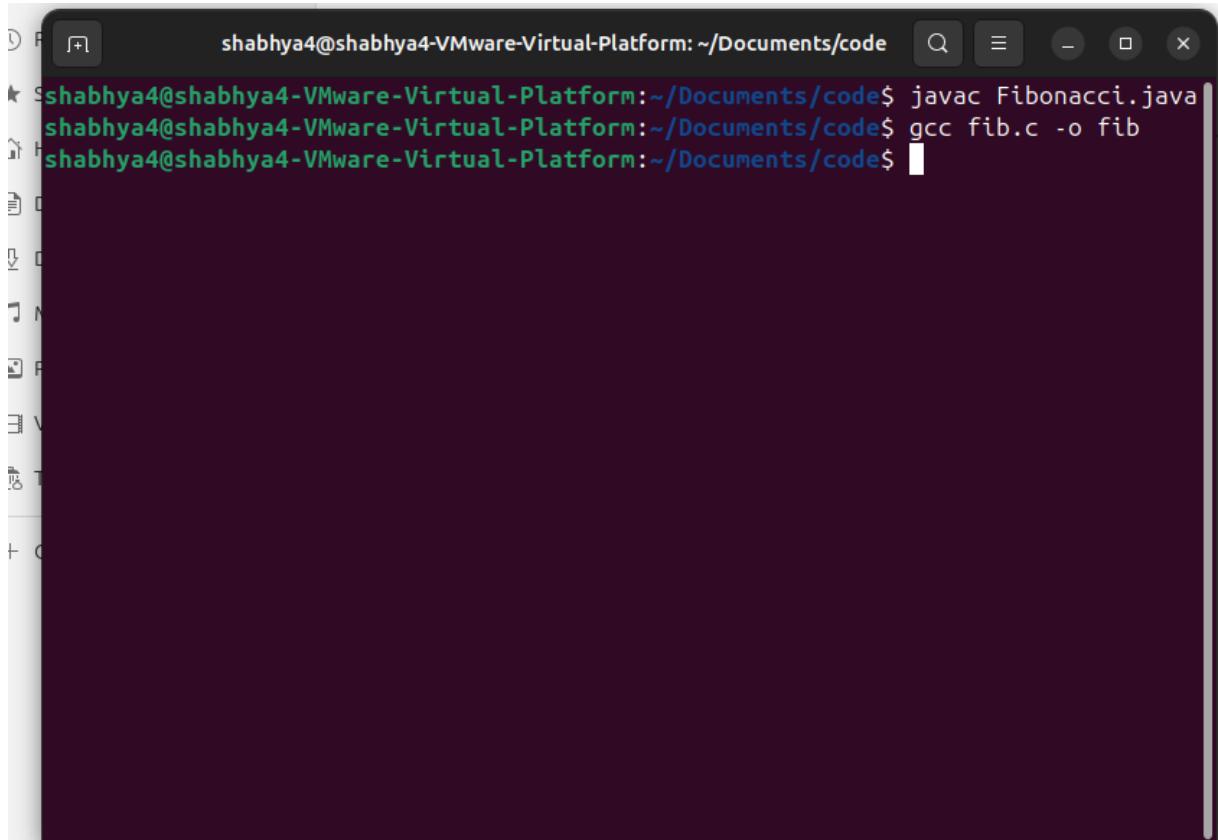
- Make executable: chmod a+x fib.sh
- Run: ./fib.sh (or bash fib.sh)

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

- Compiling Fibonacci.java creates Fibonacci. Class
- Compiling fib.c (e.g., gcc fib.c -o fib_c) creates the executable fib_

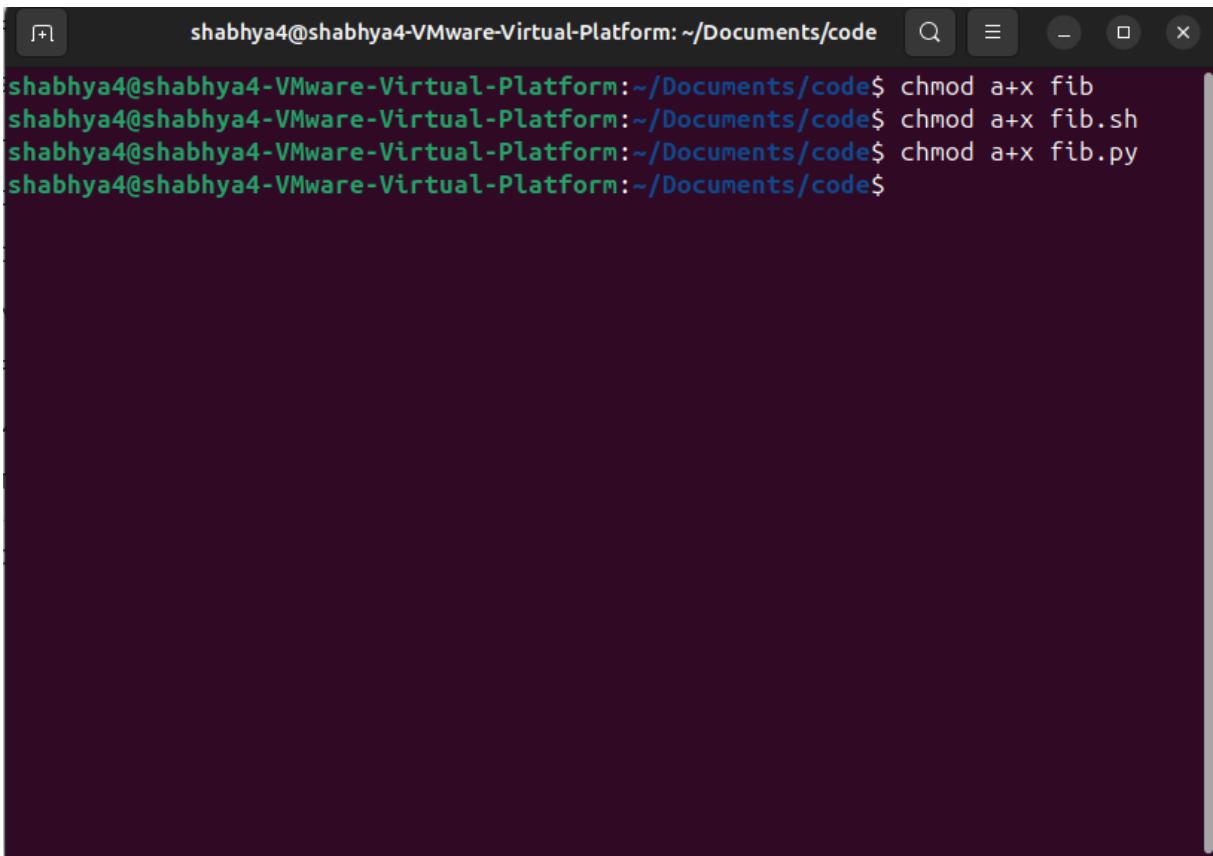
Take relevant screenshots of the following commands:

- Compile the source files where necessary



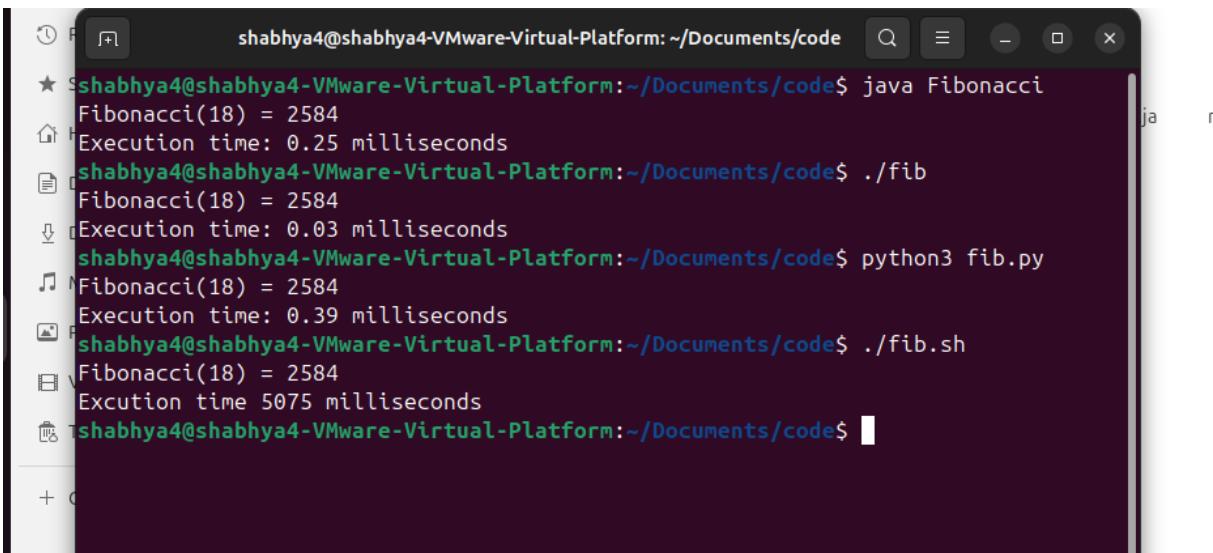
A screenshot of a terminal window titled "shabhy4@shabhy4-VMware-Virtual-Platform: ~/Documents/code". The window shows three command-line entries:
1. `javac Fibonacci.java`
2. `gcc fib.c -o fib`
3. An empty line starting with the prompt `shabhy4@shabhy4-VMware-Virtual-Platform: ~/Documents/code$`.

- Make them executable



```
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ chmod a+x fib
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ chmod a+x fib.sh
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ chmod a+x fib.py
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$
```

- Run them



```
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.25 milliseconds
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.39 milliseconds
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 5075 milliseconds
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$
```

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

After comparing the run time C programming is fastest with execution time of 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.

```
Home X Ubuntu 64-bit X
Dec 6 18:38
shabya@shabya4:~/Documents/code

-fzero-call-used-regs --param name=value -O -O0 -O1 -O2 -O3
-Ofast -Og -Oz

Program Instrumentation Options
-p -pg -fprofile-arcs --coverage -fprofile-generate
-fprofile-abs-path -fprofile-dir=path -fprofile-info-section
-fprofile-generate=path -fprofile-note=path
-fprofile-info-section=name -fprofile-update=method
-fprofile-prefix=path -fprofile-exclude-files=regex
-fprofile-filter-files=regex -fstack-protector-all
-fprofile-reproducible=[multithreaded|parallel|runs|serial]
-fsanitize=style -fsanitize-recover -fsanitize-recover=style
-fsanitize-trap -fsanitize-trap=style -fasan-shadow-offset=number
-fsanitize-sections=s1,s2,... -fsanitize-undefined-trap-on-error
-fbounds-check -fcf-protection=[full|branch|return|none|check]
-fharden-compare -fharden-conditional-branches -fstack-protector
-fstack-protector-all -fstack-protector-strong
-fstack-protector-explicit -fstack-protector-strong
-fstack-limit-registers=reg -fstack-limit-symbol=sym
-fno-stack-limit -fsplit-stack -fvtable-verify=[std|prelink|none]
-fvtv-counts -fvtv-debug -finstrument-functions
-finstrument-functions-once
-finstrument-functions-exclude-function-list=sym,sym, ...
-finstrument-functions-exclude-file-list=file,file, ...
-fprofile-prefix=old=new

Preprocessor Options
-Aquestion=answerc -Aquestion[=answer] -C -CC -Dmacro=[defn] -D
-dI -dM -dN -dU -fdebug-cpp -fdirectives-only
-fdollars-in-identifiers -fexec-charset=charset
-fextended-identifiers -finput-charset=charset
-flarge-source-files -fmacro-prefix=nasmold=new
-fmax-include-depth=depth -fno-canonical-system-headers -fpch-deps
-fpch-preprocess -fpreprocessed -ftabstop=width
-ftrack-macro-expansion -fwide-exec-charset=charset
-fworking-directory=H -fmacros file -include file -M -MD -MF
-MG -MM -MMD -MP -MQ -MT -Mno-modules -no-integrated-cpp -P
-pthread -remap -traditional -traditional-cpp -trigraphs -Umacro
-undef -Wp,option -xpreprocessor option

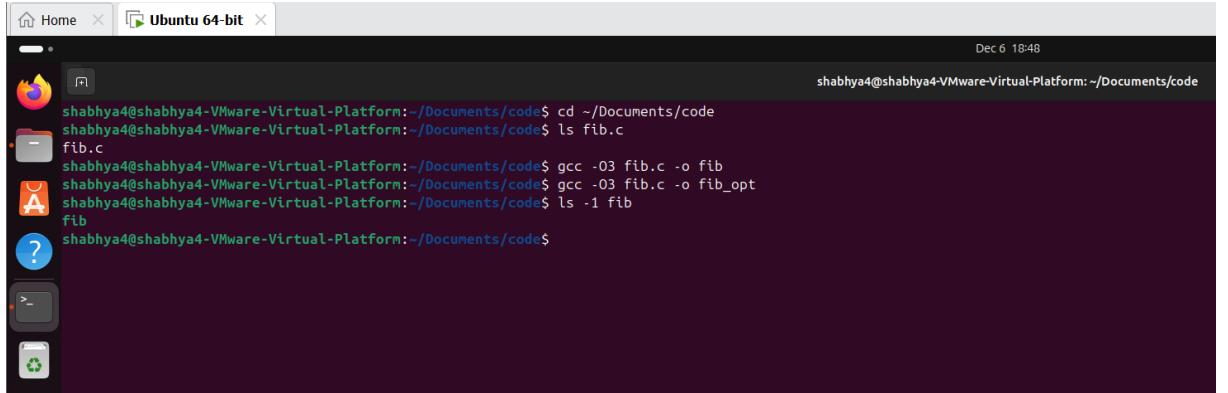
Assembler Options
-Wa,option -xassembler option

Linker Options
object-file-name -fuse-ld=linker -llibrary -nostartfiles
-nodefaultlibs -nolibc -nostdlib -nostdlib++ -e entry
-entry=entry -pie -pthread -r -rdynamic -s -static
-static-pie -static-libgcc -static-libstdc++ -static-libasan
-static-libtsan -static-libubsan -shared
-shared-libgcc -symbolic -T script -WLoption -Xlinker option -u
symbol -Z keyword

Directory Options
-Bprefix -Idir -Iidrafter dir -imacros file -imultilib dir
-iplugindir=dir -iprefix file -iquote dir -isysroot dir -l-system
dir -lwithprefix dir -lwithprefixbefore dir -Ldir
-no-canonical-prefixes --no-sysroot-suffix -nostdincl -nostdincl++
```

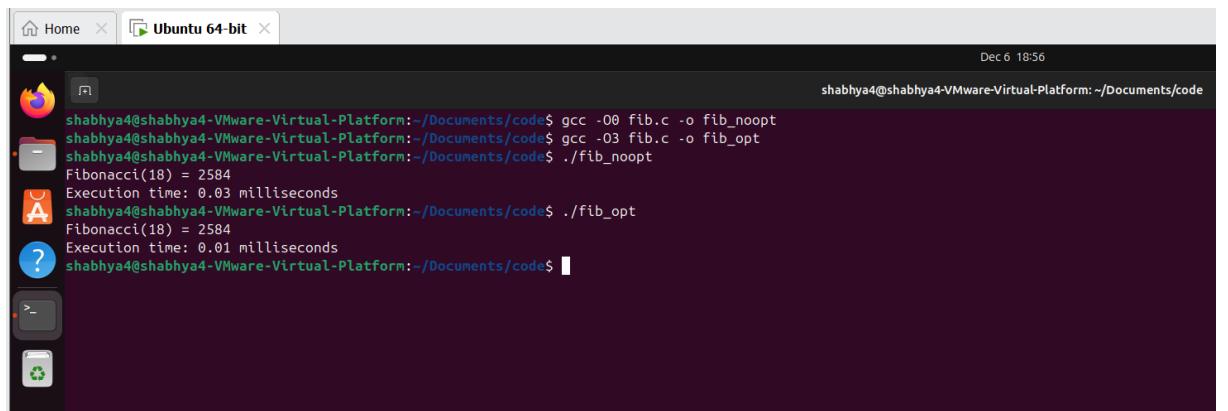
I used the `gcc` optimization flag `-O3`, found in the Optimization Options section of man `gcc`.

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



```
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ cd ~/Documents/code
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ls fib.c
fib.c
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ gcc -O3 fib.c -o fib
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ gcc -O3 fib.c -o fib_opt
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ls -l fib
fib
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$
```

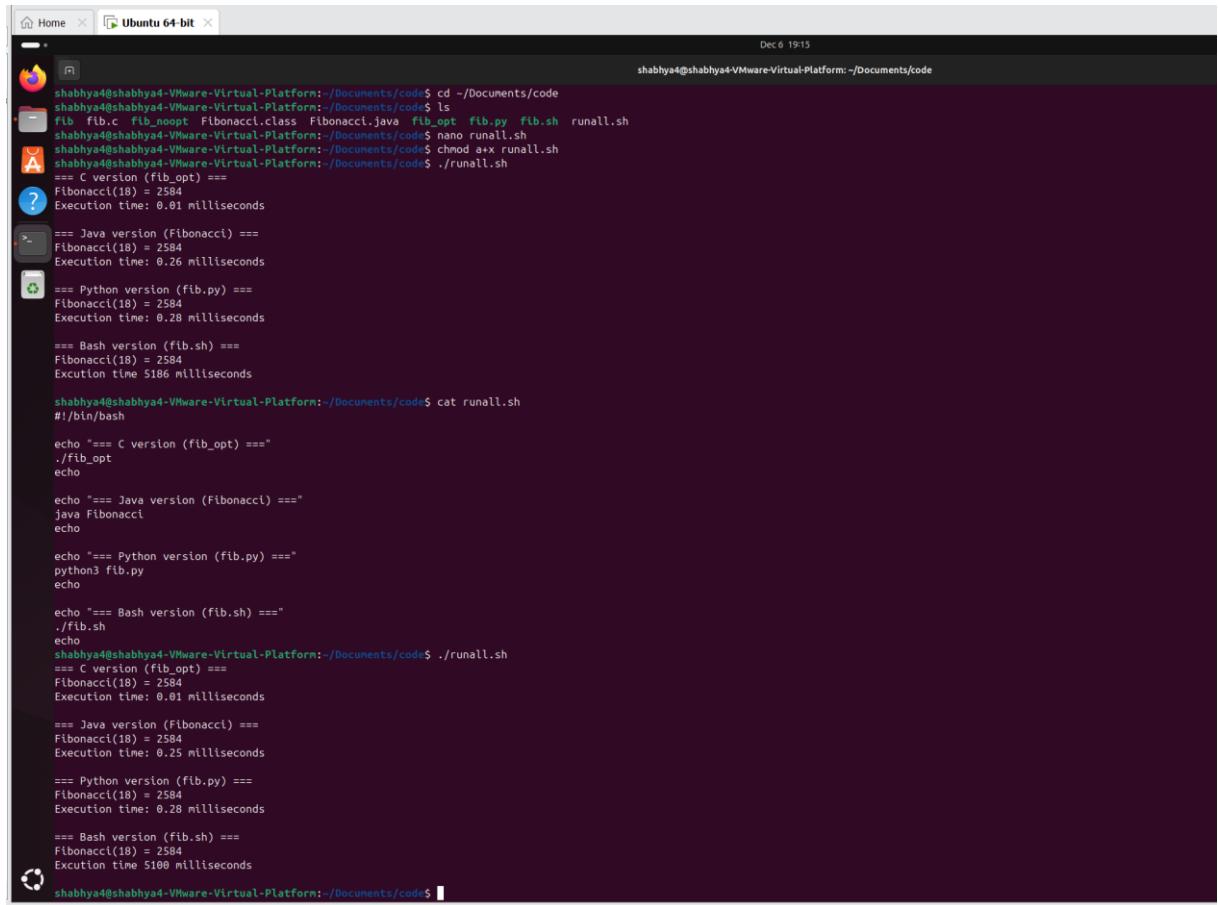
- c) Run the newly compiled program. Is it true that it now performs the calculation faster?



```
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ gcc -O0 fib.c -o fib_noopt
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ gcc -O3 fib.c -o fib_opt
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ./fib_noopt
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$
```

I compiled the C program without optimizations (-O0) and with optimizations (-O3).
The unoptimized version took about 0.03 ms; the optimized one took about 0.01 ms.
So the optimized program does run faster.

- 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.



```
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ cd ~/Documents/code
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ls
fb fb.c fb_noopt Fibonacci.class Fibonacci.java fib_opt fib.py fib.sh runall.sh
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ nano runall.sh
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ chmod a+x runall.sh
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ./runall.sh
==== C version (fib_opt) ====
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

==== Java version (Fibonacci) ====
Fibonacci(18) = 2584
Execution time: 0.26 milliseconds

==== Python version (fib.py) ====
Fibonacci(18) = 2584
Execution time: 0.28 milliseconds

==== Bash version (fib.sh) ====
Fibonacci(18) = 2584
Execution time 5186 milliseconds

shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ cat runall.sh
#!/bin/bash

echo "==== C version (fib_opt) ===="
./fib_opt
echo

echo "==== Java version (Fibonacci) ===="
java Fibonacci
echo

echo "==== Python version (fib.py) ===="
python3 fib.py
echo

echo "==== Bash version (fib.sh) ===="
./fib.sh
echo
shabhy4@shabhy4-VMware-Virtual-Platform:~/Documents/code$ ./runall.sh
==== C version (fib_opt) ====
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

==== Java version (Fibonacci) ====
Fibonacci(18) = 2584
Execution time: 0.25 milliseconds

==== Python version (fib.py) ====
Fibonacci(18) = 2584
Execution time: 0.28 milliseconds

==== Bash version (fib.sh) ====
Fibonacci(18) = 2584
Execution time 5100 milliseconds
```

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, #1
```

Loop:

```
mul r0, r0, r1
subs r2, r2, #1
bne 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. The assembly code is displayed in the left pane, and the register and memory states are shown in the right pane.

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     subs r2, r2, #1
9     bne Loop
10
11 End:
```

Registers:

Register	Value
R0	10
R1	2
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
SP	10000

Memory Dump:

Address	Value
0x00010000	02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 90 01 00 E0 .. R
0x00010010	01 20 52 E2 FC FF FF 1A 00 00 00 00 00 00 00 00 ..
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 ..

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