

Week 4 – Software

Student number: 578438

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

```
1 Main:
2   mov r2, #5
3   mov r1, #1
4
5 loop:
6   cmp r2, #1
7   beq End
8   mul r1, r1, r2
9   sub r2, r2, #1
10  b loop
11
12 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
R11	0
R12	0
SP	10000

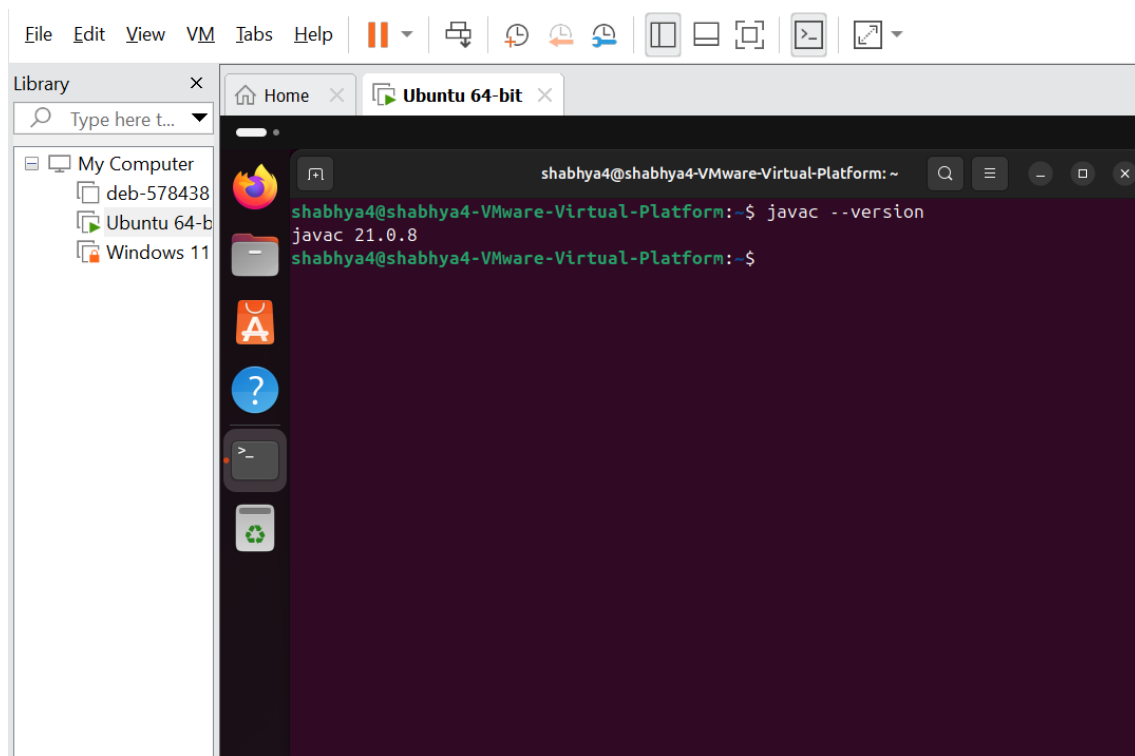
Memory dump (hex):

```
0x00010000: 05 20 A0 E3 01 10 A0 E3 01 00 52 E3 02 00 00 0A ... R
0x00010010: 91 02 01 E0 01 20 42 E2 FA FF FF EA ... B
0x00010020: 10 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 ...
```

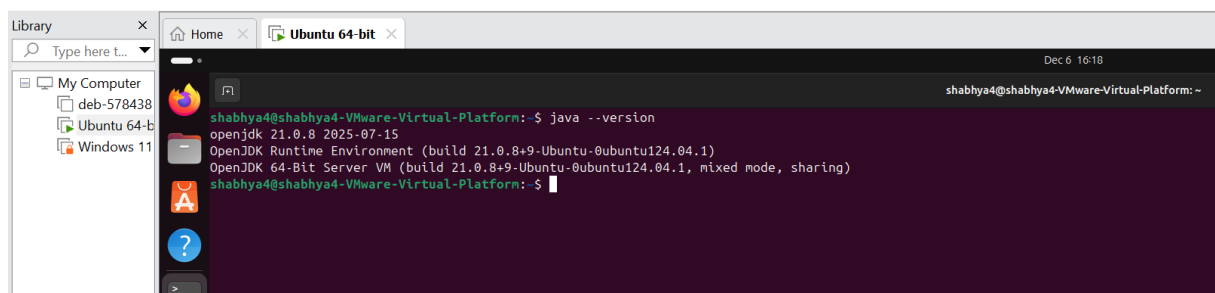
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

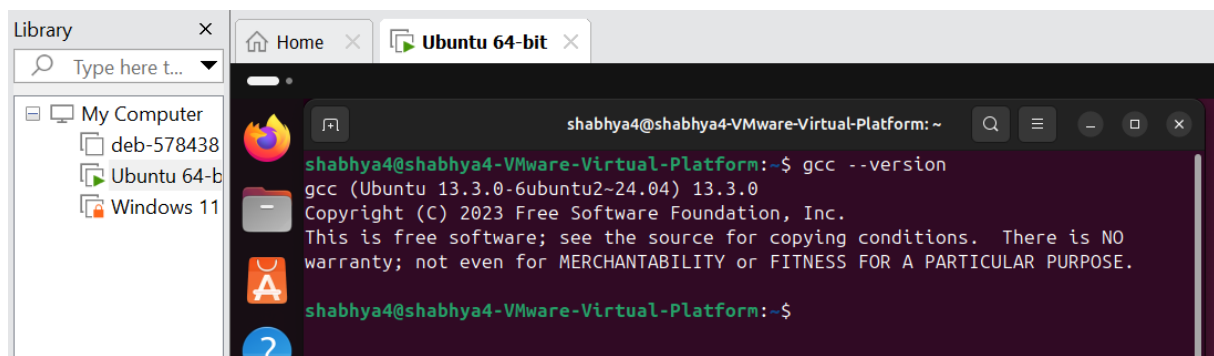
`javac --version`



`java --version`



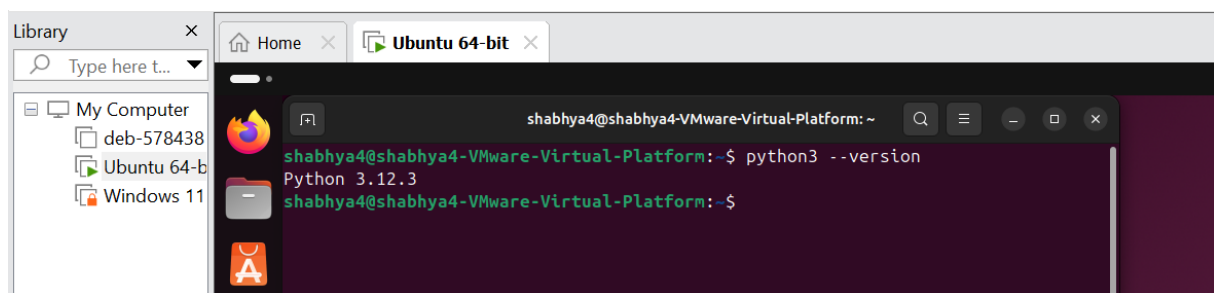
gcc --version



The screenshot shows a terminal window titled 'shabhya4@shabhya4-VMware-Virtual-Platform: ~'. The command 'gcc --version' has been executed, resulting in the following output:

```
shabhya4@shabhya4-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.
shabhya4@shabhya4-VMware-Virtual-Platform:~$
```

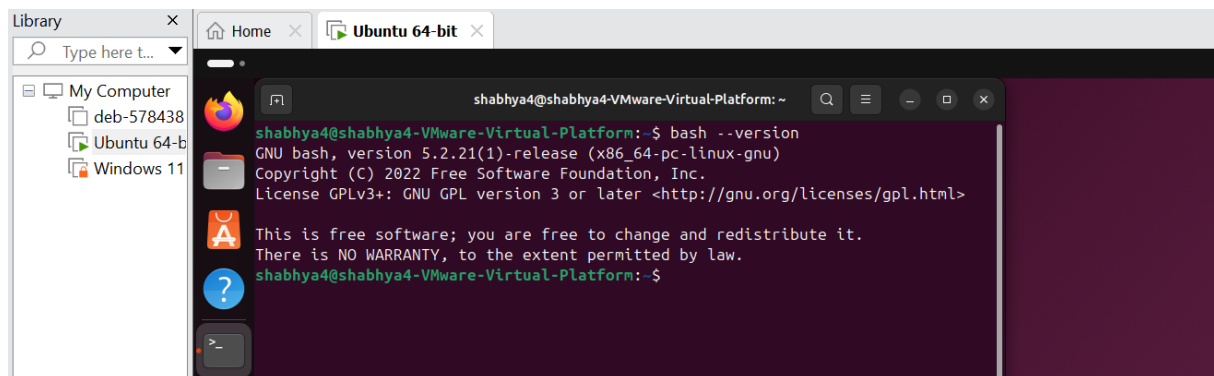
python3 --version



The screenshot shows a terminal window titled 'shabhya4@shabhya4-VMware-Virtual-Platform: ~'. The command 'python3 --version' has been executed, resulting in the following output:

```
shabhya4@shabhya4-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
shabhya4@shabhya4-VMware-Virtual-Platform:~$
```

bash --version



The screenshot shows a terminal window titled 'shabhya4@shabhya4-VMware-Virtual-Platform: ~'. The command 'bash --version' has been executed, resulting in the following output:

```
shabhya4@shabhya4-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.
shabhya4@shabhya4-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?

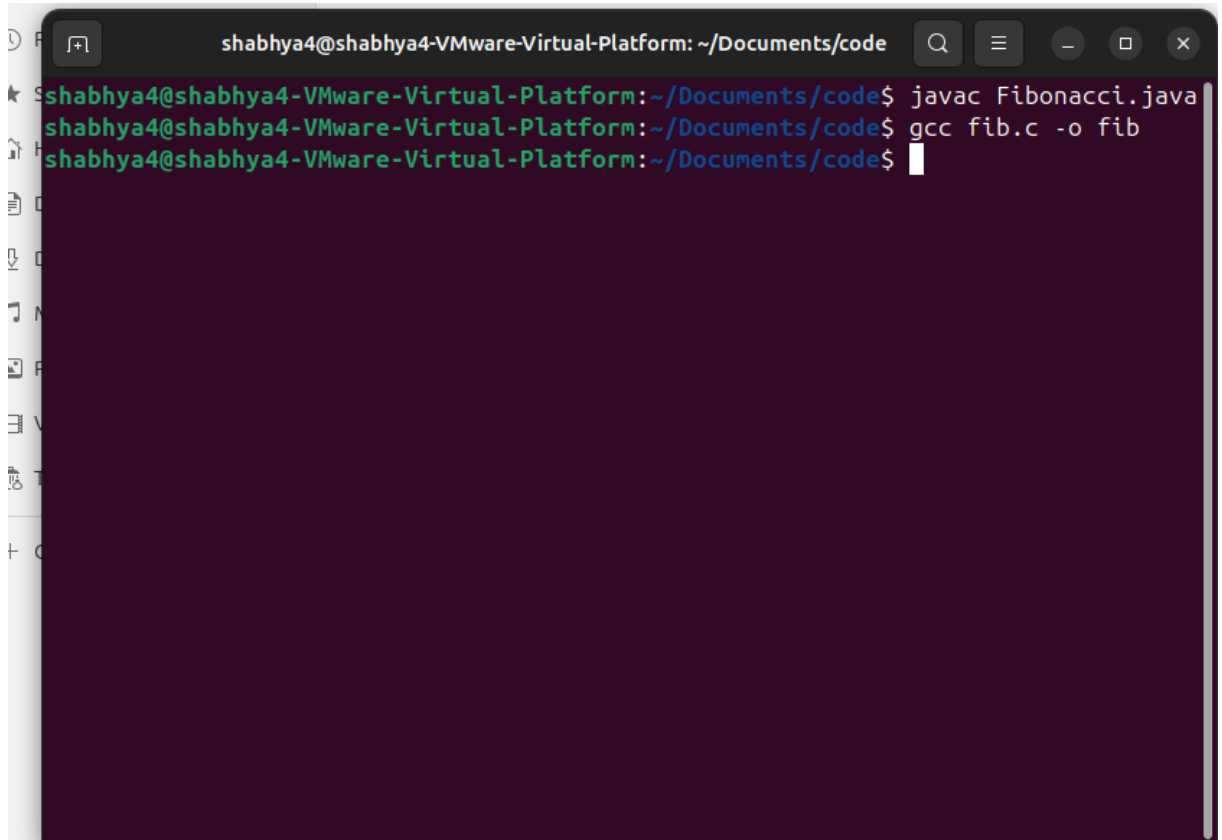
- Make executable: `chmod a+x fib.sh`
- Run: `./fib.sh` (or `bash fib.sh`)

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_

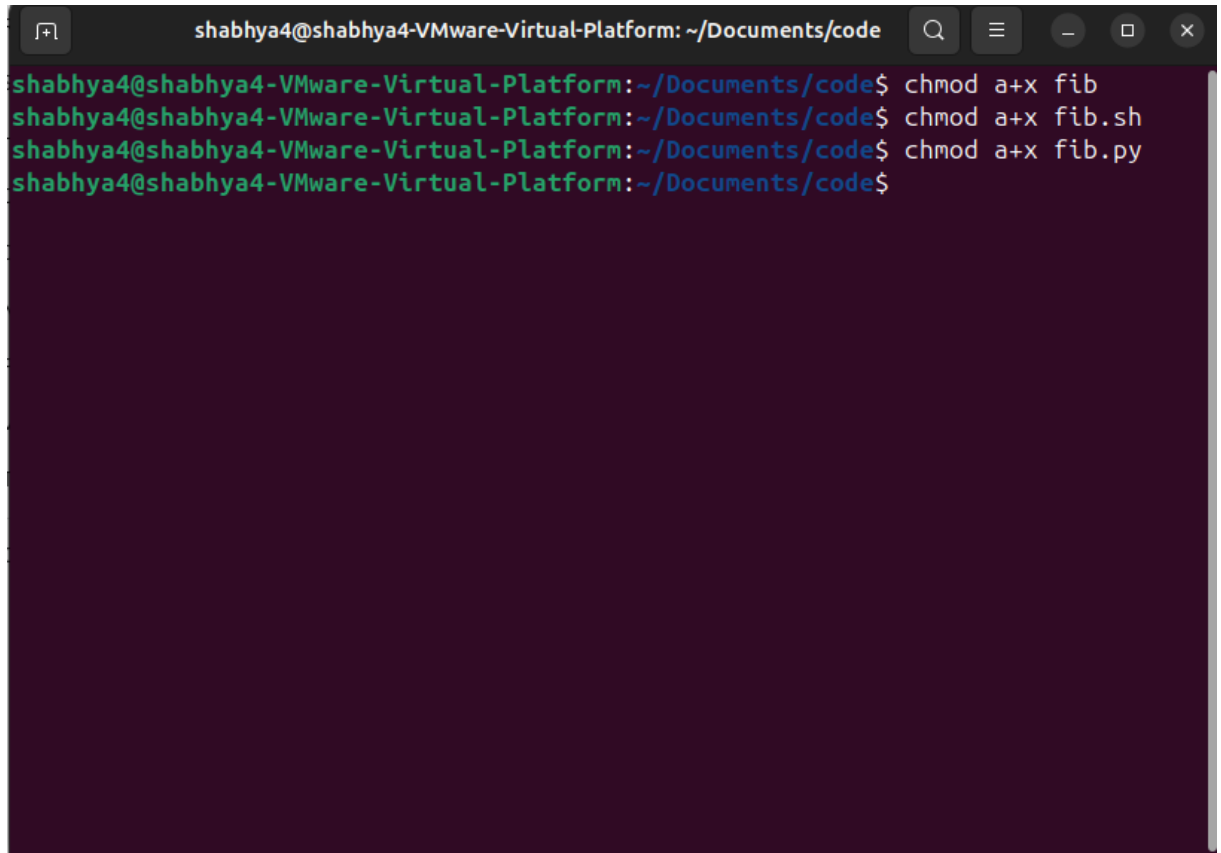
Take relevant screenshots of the following commands:

- Compile the source files where necessary

A screenshot of a terminal window with a dark purple background. The window title is 'shabhya4@shabhya4-VMware-Virtual-Platform: ~/Documents/code'. The terminal shows three lines of commands entered at the prompt: 'javac Fibonacci.java', 'gcc fib.c -o fib', and a third prompt line with a cursor. The left sidebar of the terminal shows various icons for file management and development tools.

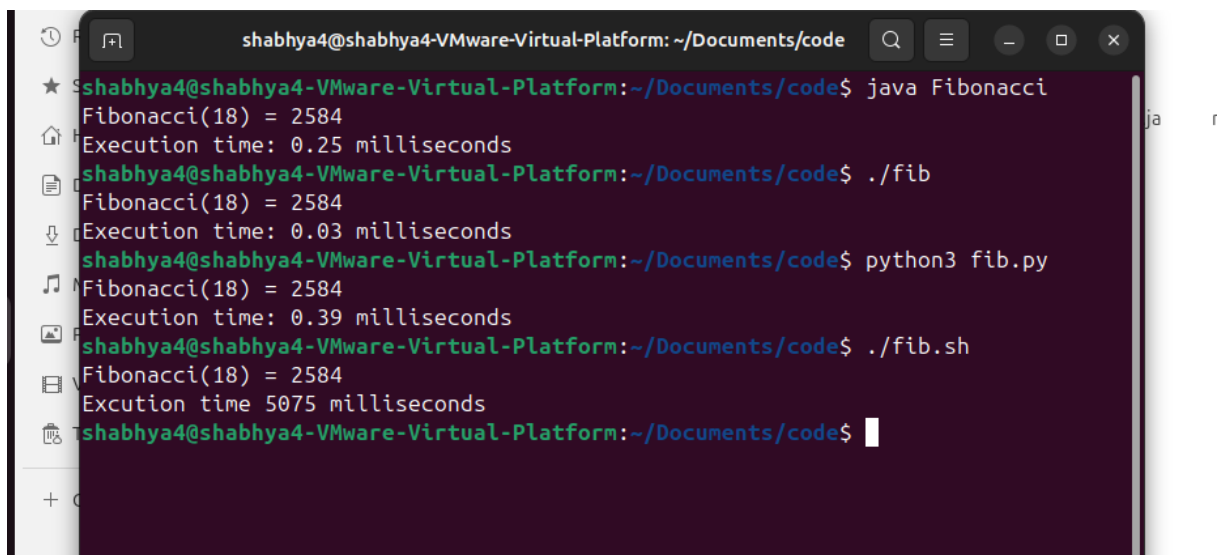
```
shabhya4@shabhya4-VMware-Virtual-Platform: ~/Documents/code
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ javac Fibonacci.java
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ gcc fib.c -o fib
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$
```

- Make them executable



```
shabhya4@shabhya4-VMware-Virtual-Platform: ~/Documents/code
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ chmod a+x fib
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ chmod a+x fib.sh
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ chmod a+x fib.py
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$
```

- Run them



```
shabhya4@shabhya4-VMware-Virtual-Platform: ~/Documents/code
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.25 milliseconds
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.39 milliseconds
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 5075 milliseconds
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$
```

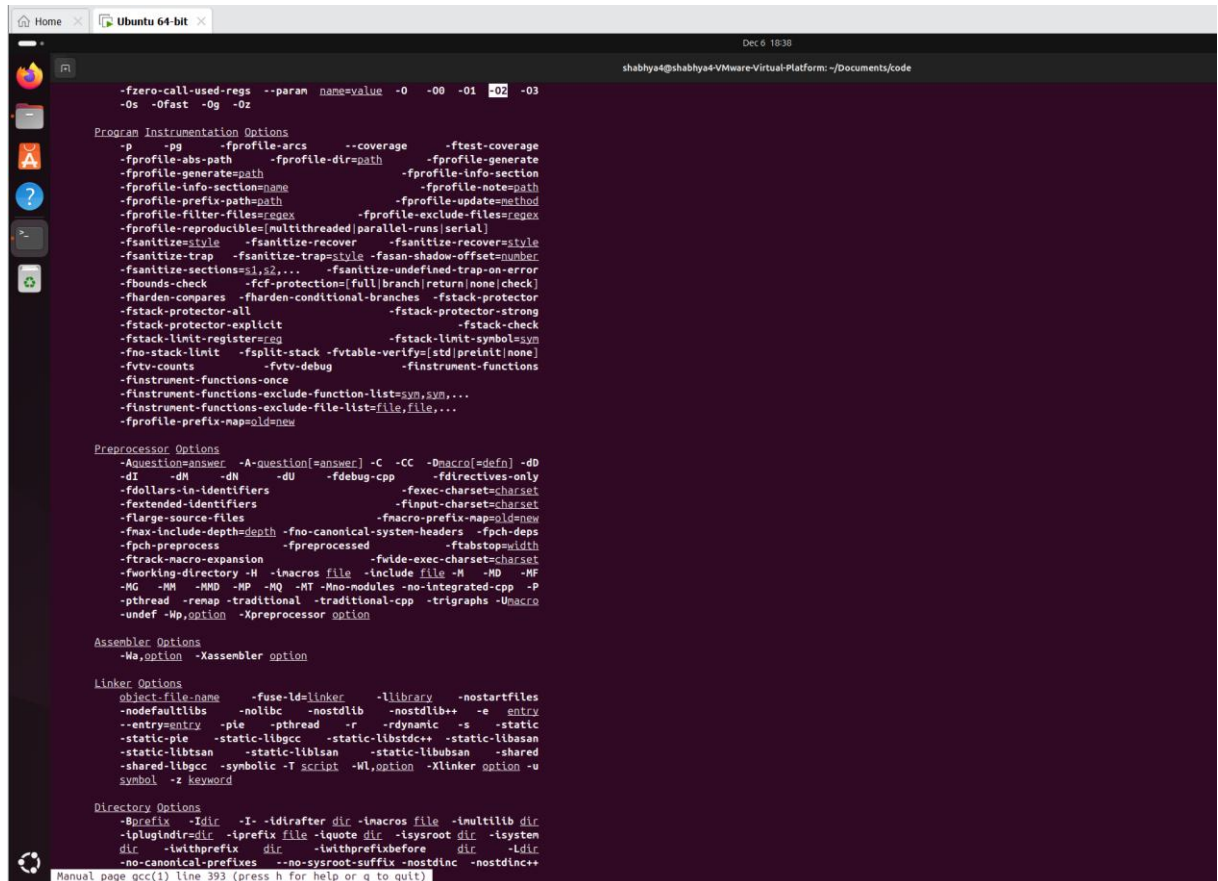
- Which (compiled) source code file performs the calculation the fastest?

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:

- 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.



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

Preprocessor Options
-Aquestion=answer -Aquestion[=answer] -C -CC -Dmacro[=defn] -dD
-dl -dM -dM -dU -fdebug-cpp -fdirectives-only
-fdollars-in-identifiers -fexec-charset=charset
-fextended-identifiers -finput-charset=charset
-flarge-source-files -fmacro-prefix-map=old=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 -Iacros file -include file -H -MD -MF
-MG -MW -MMD -MP -MQ -MT -Mno-modules -no-integrated-cpp -P
-pthread -renap-traditional -traditional-cpp -trigraphs -Umacro
-undef -Wp,option -Xpreprocessor option

Assembler Options
-Wa,option -Xassembler option

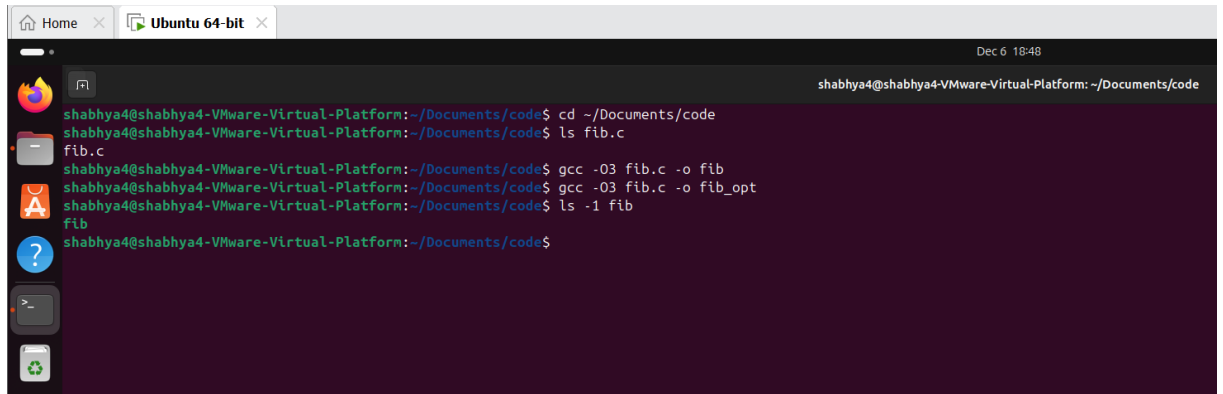
Linker Options
object-file-name -fuse-ld=linker -llibrary -nostartfiles
-nodfaulthlib -nolibc -nostdlib -nostdlib++ -e entry
--entry=entry -pie -pthread -r -rdynamic -s -static
-static-pie -static-libgcc -static-libstdc++ -static-libasan
-static-libtsan -static-liblsan -static-libubsan -shared
-shared-libgcc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

Directory Options
-Bprefix -Idir -I- -idirafter dir -inacros file -multilib dir
-lplugin=dir -iprefix file -iquote dir -isysroot dir -isystem
dir -lwithprefix dir -lwithprefixbefore dir -Ldir
-no-canonical-prefixes --no-sysroot-suffix -nostdinc -nostdinc++

Manual page gcc(1) line 393 (press h for help or q to quit)
```

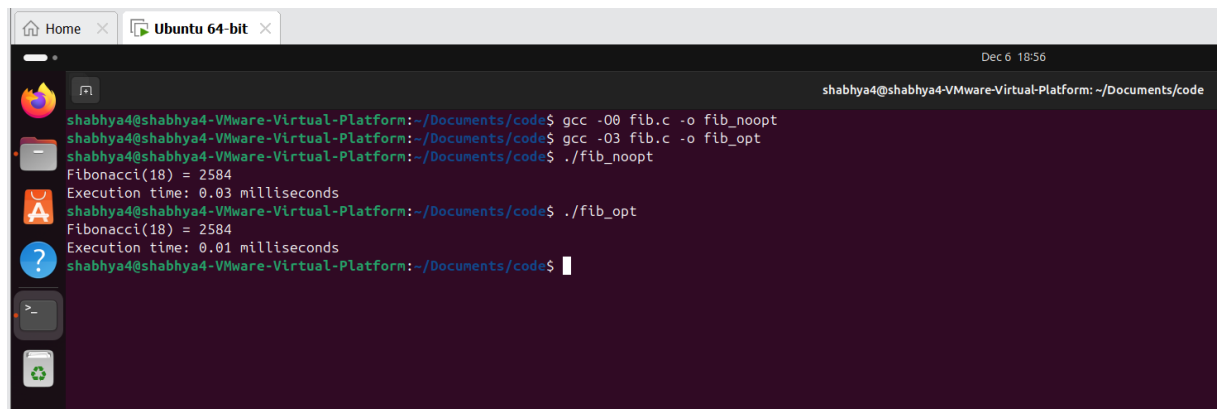
I used the gcc optimization flag -O3, found in the Optimization Options section of man gcc.

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



```
shabhya4@shabhya4-VMware-Virtual-Platform: ~/Documents/code
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ cd ~/Documents/code
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ ls fib.c
fib.c
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ gcc -O3 fib.c -o fib
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ gcc -O3 fib.c -o fib_opt
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ ls -l fib
-rwxr-xr-x 1 shabhya4 shabhya4 12288 Dec 6 18:48 fib
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$
```

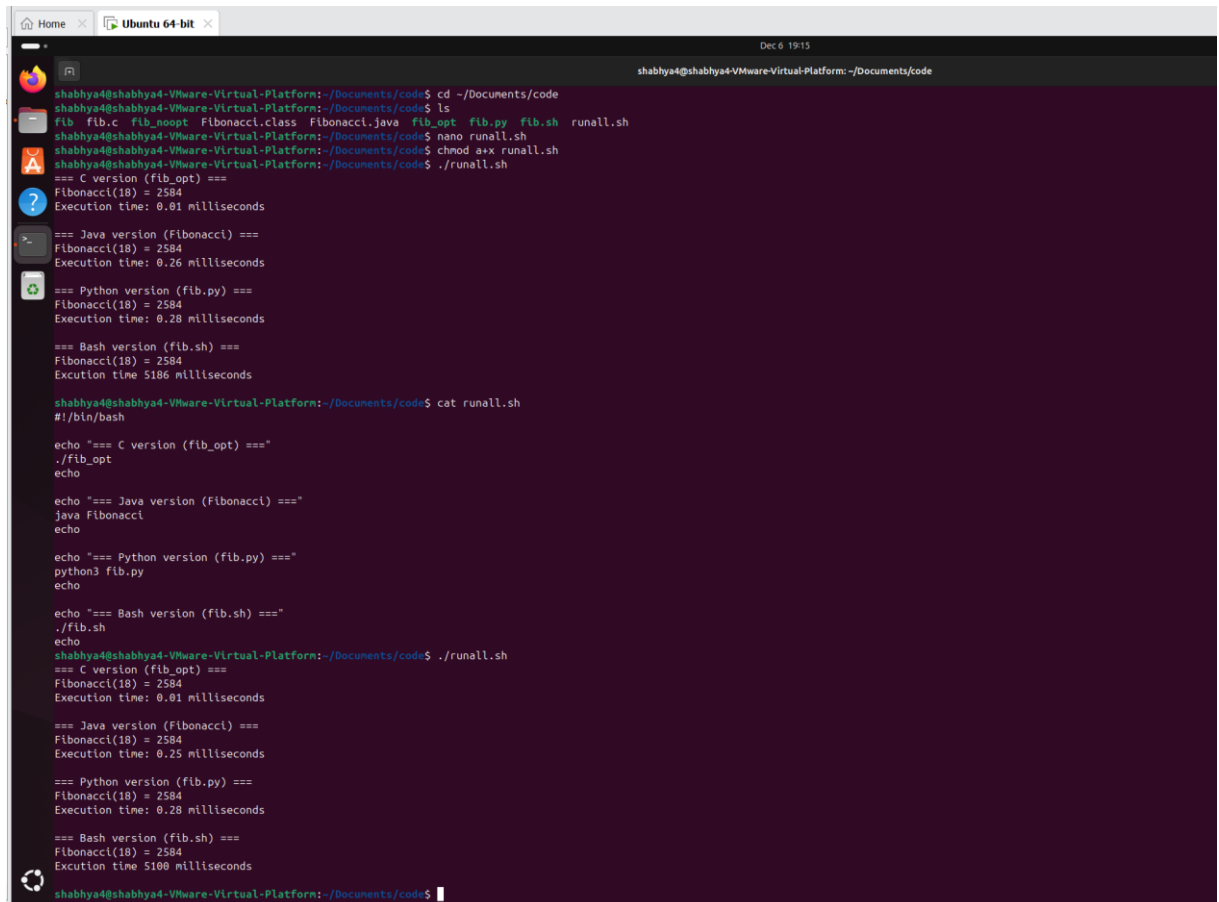
- c) Run the newly compiled program. Is it true that it now performs the calculation faster?



```
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ gcc -O0 fib.c -o fib_noopt
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ gcc -O3 fib.c -o fib_opt
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ ./fib_noopt
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
shabhya4@shabhya4-VMware-Virtual-Platform:~/Documents/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
shabhya4@shabhya4-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-Virtual-Platform:~/Documents/code$ cd ~/Documents/code
shabhy4@shabhy4-Virtual-Platform:~/Documents/code$ ls
fib fib.c fib_noopt Fibonacci.class Fibonacci.java fib_opt fib.py fib.sh runall.sh
shabhy4@shabhy4-Virtual-Platform:~/Documents/code$ nano runall.sh
shabhy4@shabhy4-Virtual-Platform:~/Documents/code$ chmod a+x runall.sh
shabhy4@shabhy4-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-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-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

shabhy4@shabhy4-Virtual-Platform:~/Documents/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, #1
```

Loop:

```
mul r0, r0, r1
```

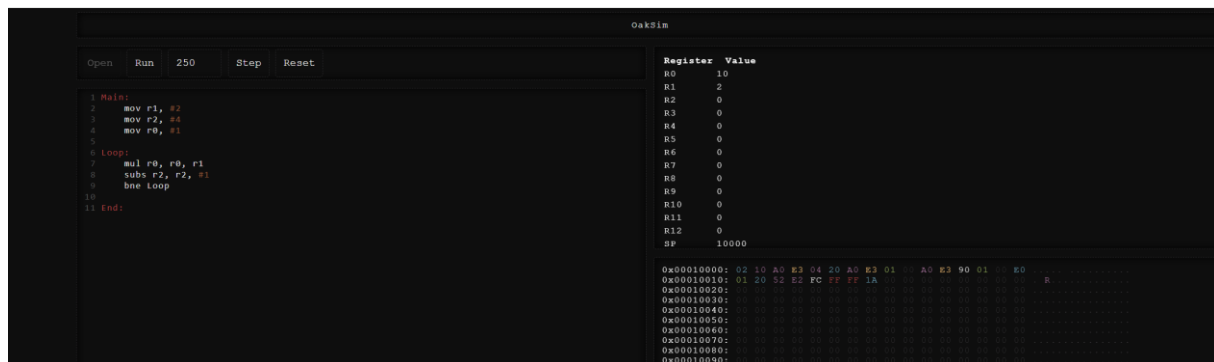
```
subs r2, r2, #1
```

```
bne Loop
```

End:

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

Screenshot of the completed code here.



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