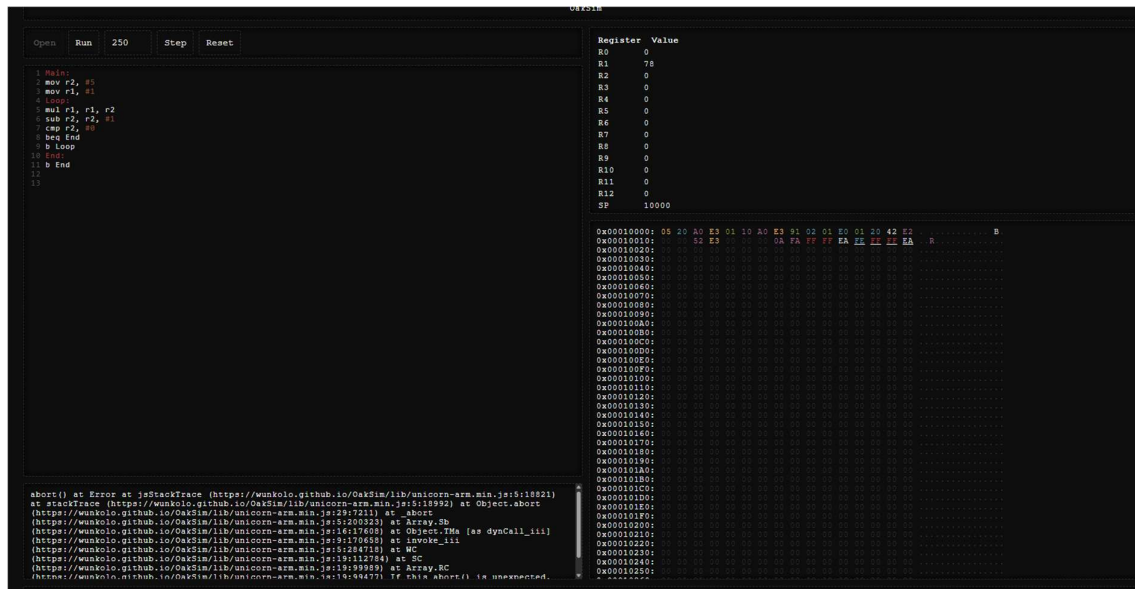


Template Week 4 – Software

Student number: 575798

Assignment 4.1: ARM assembly

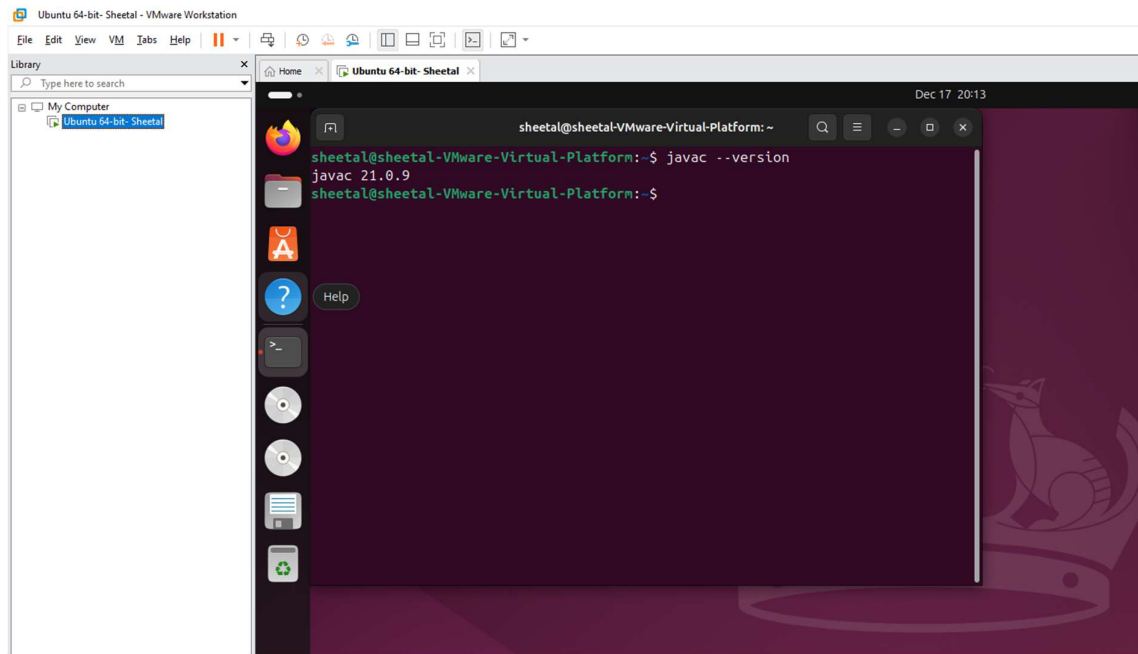
Screenshot of working assembly code of factorial calculation:



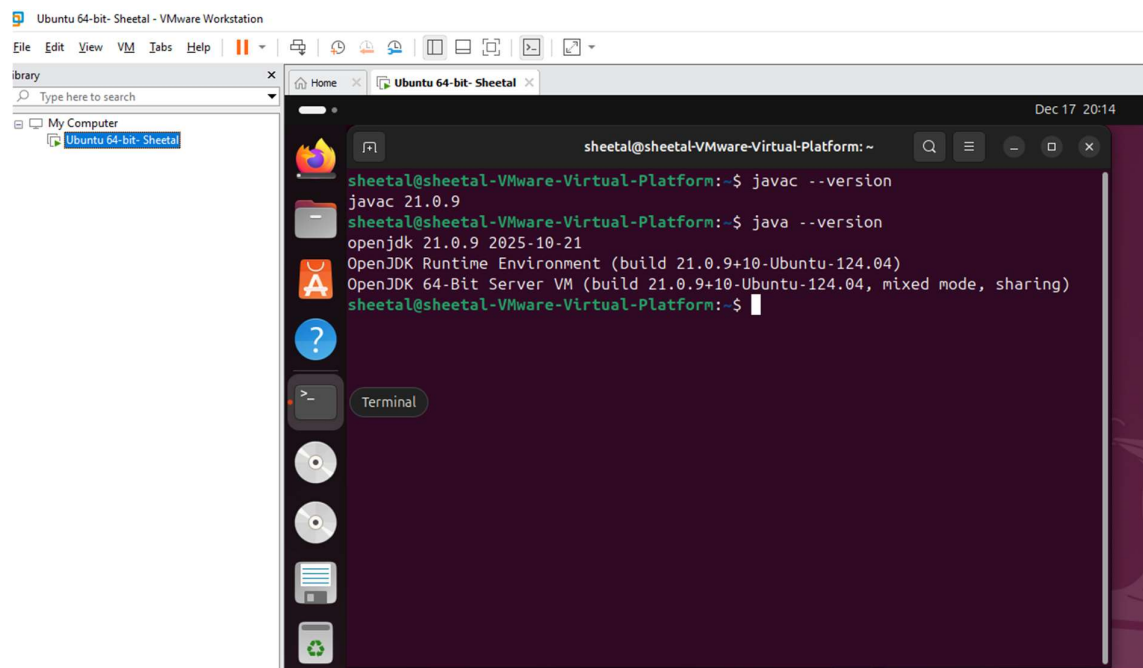
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

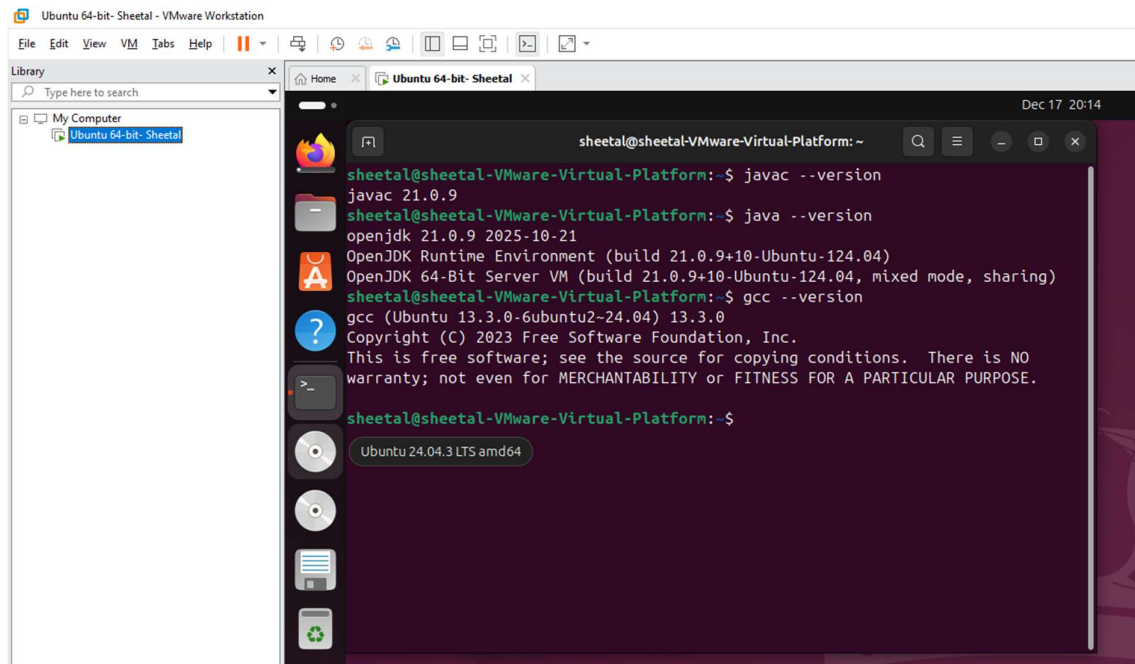
```
javac -version
```



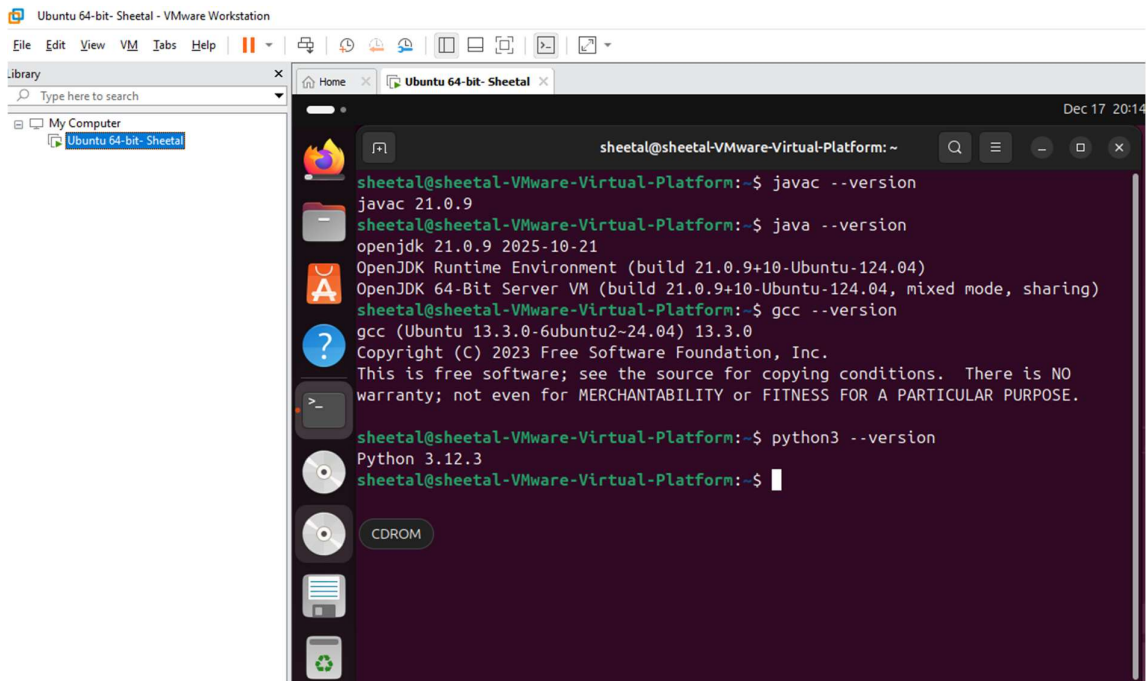
java --version



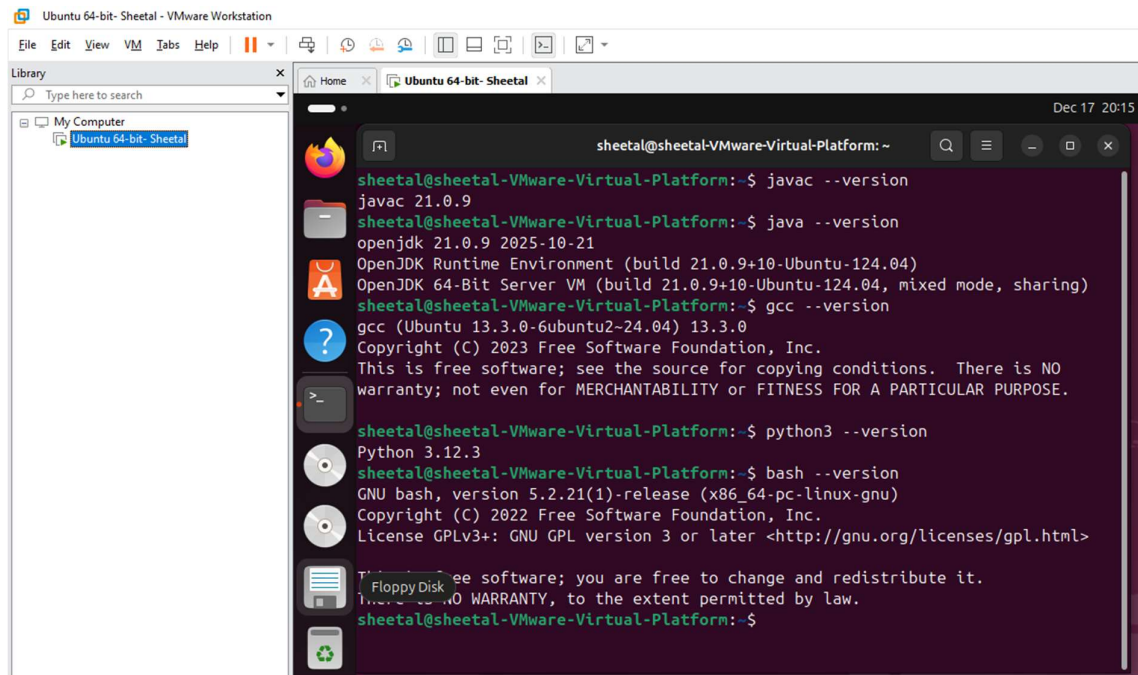
gcc --version



python3 --version



bash --version



Assignment 4.3: Compile

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

The C source file needs to be compiled. The java source file needs to be compiled. The python file and bash script does not need to be compiled.

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

The C compiler (gcc) compiles C code directly into native machine code, which is executed directly by the CPU.

Which source code files are compiled to byte code?

Java source code is compiled by java c into Java bytecode which runs on the Java Virtual Machine.

Which source code files are interpreted by an interpreter?

Python (.py) interpreted by the Python interpreter

Bash (.sh) interpreted by the Bash shell

These source code files will perform the same calculation after compilation/interpretation. Which one is expected to do the calculation the fastest?

C program is expected to be the fastest since it is compiled into native machine code. Also, because there is no virtual machine or interpreter overhead and it runs directly on the CPU.

How do I run a Java program?

Compile javac Fib.java

How do I run a Python program?

python3 fib.py

How do I run a C program?

gcc fib.c -o fib

How do I run a Bash script?

sudo chmod a+x fib.sh

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

For java there will be a bytecode file and for C there will be an executable file. For python and bash there will be no compiled file.

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?

```
sheetal@sheetal-VMware-Virtual-Platform: ~/code
sheetal@sheetal-VMware-Virtual-Platform:~$ gcc fib.c -o fib
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetal@sheetal-VMware-Virtual-Platform:~$ cd code
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
sheetal@sheetal-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib
javac Fibonacci.java
sheetal@sheetal-VMware-Virtual-Platform:~/code$ chmod a+x fib
sudo chmod a+x fib.sh
[sudo] password for sheetal:
sheetal@sheetal-VMware-Virtual-Platform:~/code$
```

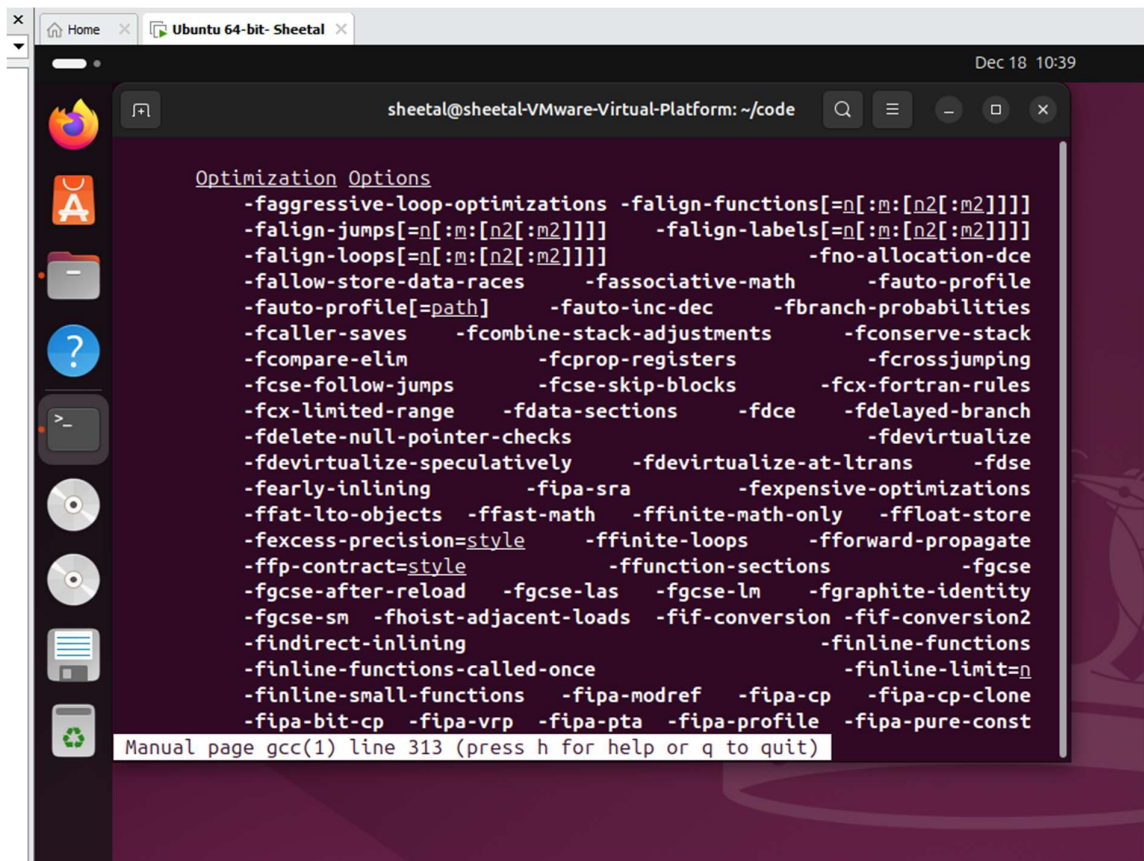
```
sheetal@sheetal-VMware-Virtual-Platform: ~/code
sheetal@sheetal-VMware-Virtual-Platform:~$ gcc fib.c -o fib
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetal@sheetal-VMware-Virtual-Platform:~$ cd code
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
sheetal@sheetal-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib
javac Fibonacci.java
sheetal@sheetal-VMware-Virtual-Platform:~/code$ chmod a+x fib
sudo chmod a+x fib.sh
[sudo] password for sheetal:
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ./fib
java Fibonacci
python3 fib.py
sudo ./fib.sh
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
Fibonacci(18) = 2584
Execution time: 0.19 milliseconds
Fibonacci(18) = 2584
Execution time: 0.24 milliseconds
Fibonacci(18) = 2584
Execution time 4758 milliseconds
sheetal@sheetal-VMware-Virtual-Platform:~/code$
```


C calculates the fastest.

Assignment 4.4: Optimize

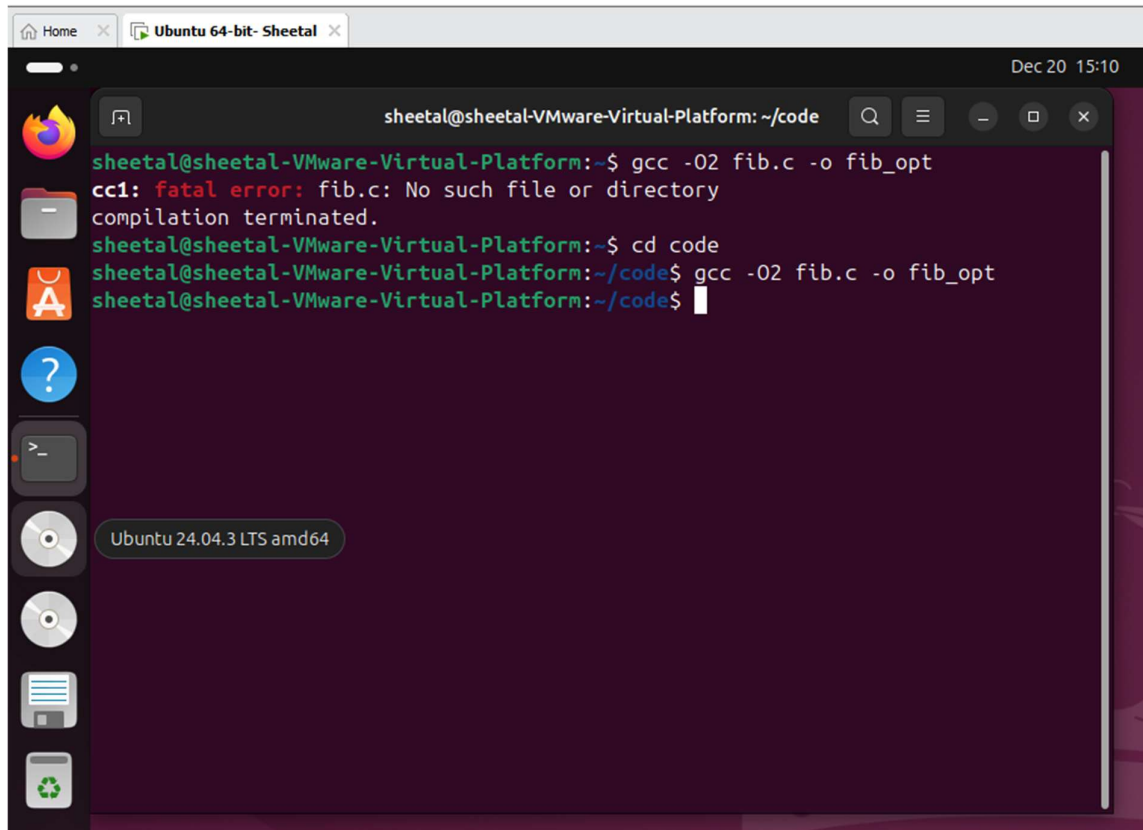
Take relevant screenshots of the following commands:

- 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.

A screenshot of a terminal window titled 'sheetal@sheetal-VMware-Virtual-Platform: ~/code'. The terminal displays a list of GCC optimization options under the heading 'Optimization Options'. The options are listed in multiple columns, including flags like -faggressive-loop-optimizations, -falign-functions, -falign-jumps, -falign-labels, -falign-loops, -fno-allocation-dce, -fallow-store-data-races, -fassociative-math, -fauto-profile, -fauto-profile[=path], -fauto-inc-dec, -fbranch-probabilities, -fcaller-saves, -fcombine-stack-adjustments, -fconserve-stack, -fcompare-elim, -fcprop-registers, -fcrossjumping, -fcse-follow-jumps, -fcse-skip-blocks, -fcx-fortran-rules, -fcx-limited-range, -fddata-sections, -fdce, -fdelayed-branch, -fdelete-null-pointer-checks, -fdevirtualize, -fdevirtualize-speculatively, -fdevirtualize-at-ltrans, -fdse, -fearly-inlining, -fipa-sra, -fexpensive-optimizations, -ffat-lto-objects, -ffast-math, -ffinite-math-only, -ffloat-store, -fexcess-precision=style, -ffinite-loops, -fforward-propagate, -ffp-contract=style, -ffunction-sections, -fgcse, -fgcse-after-reload, -fgcse-las, -fgcse-lm, -fgraphite-identity, -fgcse-sm, -fhoist-adjacent-loads, -fif-conversion, -fif-conversion2, -findirect-inlining, -finline-functions, -finline-functions-called-once, -finline-limit=n, -finline-small-functions, -fipa-modref, -fipa-cp, -fipa-cp-clone, -fipa-bit-cp, -fipa-vrp, -fipa-pta, -fipa-profile, and -fipa-pure-const. At the bottom of the terminal, a message reads 'Manual page gcc(1) line 313 (press h for help or q to quit)'. The terminal window has a sidebar with various application icons on the left and a top bar with window controls and the current directory path.

I can see that gcc supports multiple optimisation flags. The recommended optimisation level is -O2 because there is a wider range of optimisations that can increase the speed.

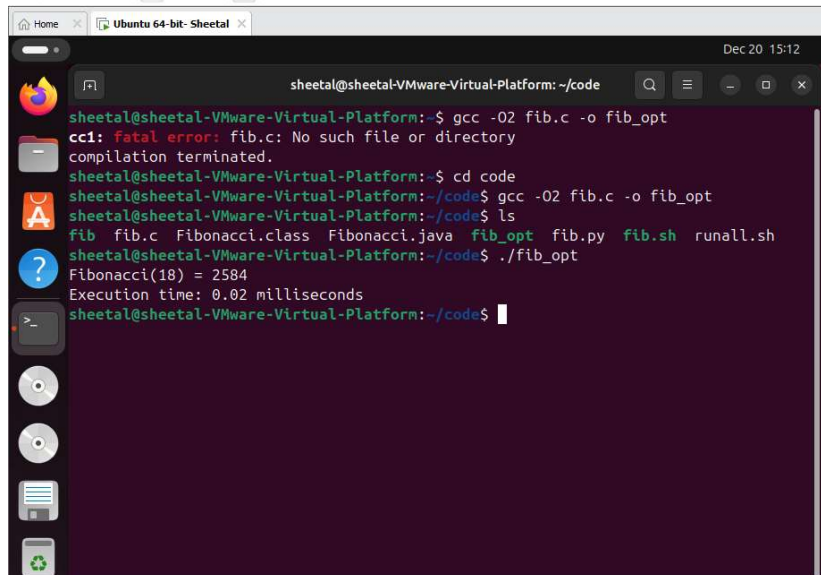
- Compile **fib.c** again with the optimization parameters



The terminal window shows the user attempting to compile a C program. The first command fails due to a missing file, and the second succeeds after navigating to the correct directory.

```
sheetal@sheetal-VMware-Virtual-Platform: ~/code
sheetal@sheetal-VMware-Virtual-Platform:~$ gcc -O2 fib.c -o fib_opt
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetal@sheetal-VMware-Virtual-Platform:~$ cd code
sheetal@sheetal-VMware-Virtual-Platform:~/code$ gcc -O2 fib.c -o fib_opt
sheetal@sheetal-VMware-Virtual-Platform:~/code$
```

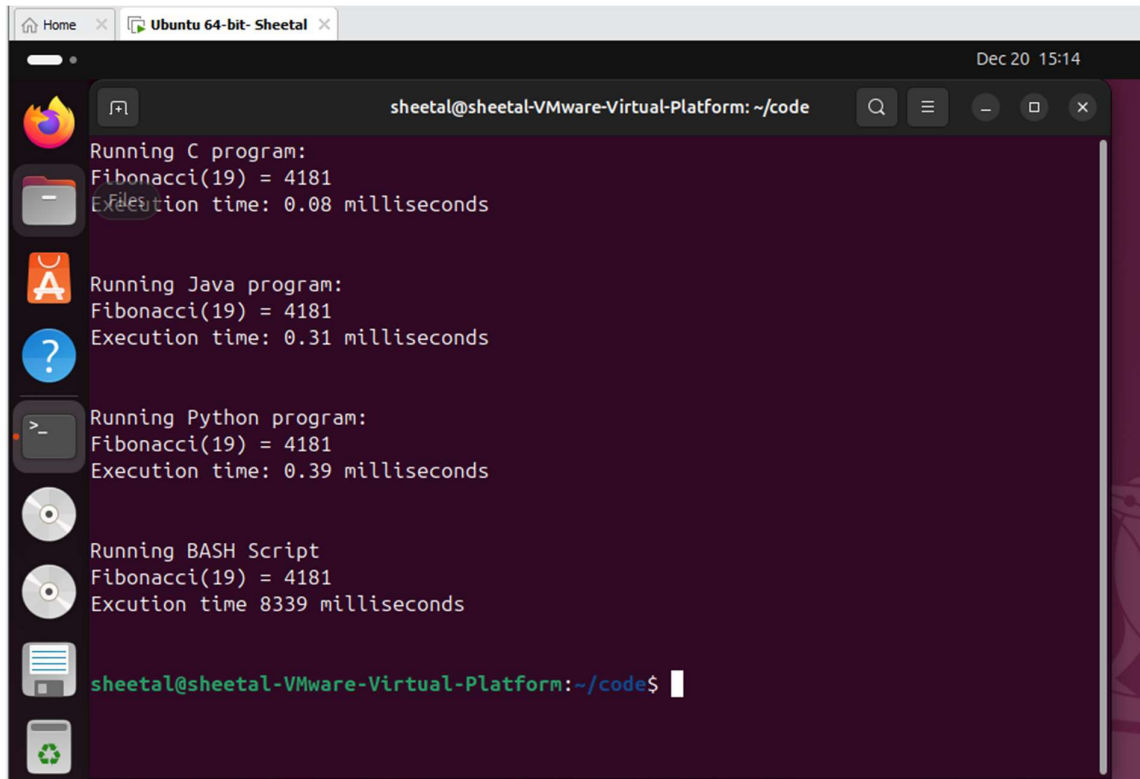
c) Run the newly compiled program. Is it true that it now performs the calculation faster?



The terminal window shows the user running the compiled program and listing the files in the directory. The program outputs the 18th Fibonacci number and its execution time.

```
sheetal@sheetal-VMware-Virtual-Platform: ~/code
sheetal@sheetal-VMware-Virtual-Platform:~$ gcc -O2 fib.c -o fib_opt
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetal@sheetal-VMware-Virtual-Platform:~$ cd code
sheetal@sheetal-VMware-Virtual-Platform:~/code$ gcc -O2 fib.c -o fib_opt
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib_opt fib.py fib.sh runall.sh
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
sheetal@sheetal-VMware-Virtual-Platform:~/code$
```


- 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.



```
sheetal@sheetal-VMware-Virtual-Platform: ~/code
Running C program:
Fibonacci(19) = 4181
Execution time: 0.08 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.31 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.39 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 8339 milliseconds

sheetal@sheetal-VMware-Virtual-Platform: ~/code$
```

As we can see C program performs the calculations the fastest.

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
```

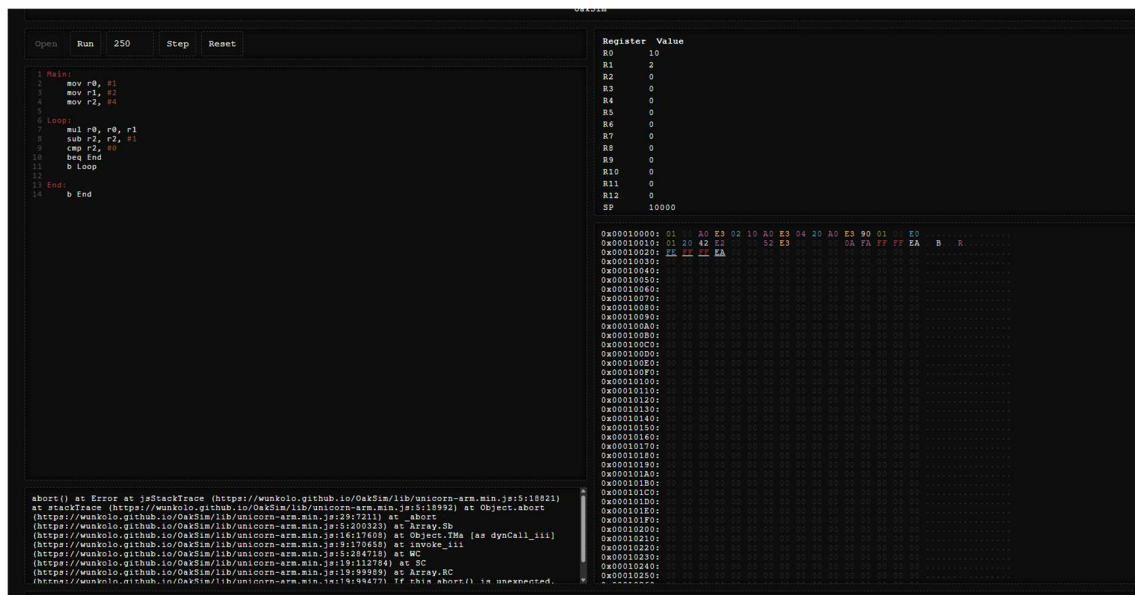
```
mov r2, #4
```

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**