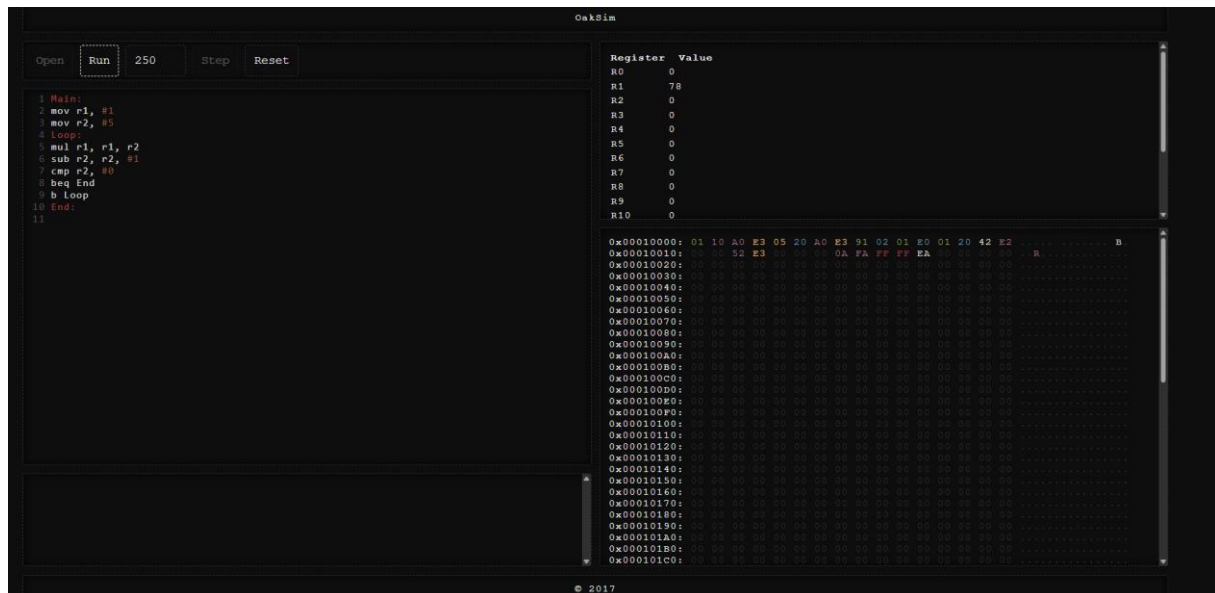


Template Week 4 – Software

Student number: 592801

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:

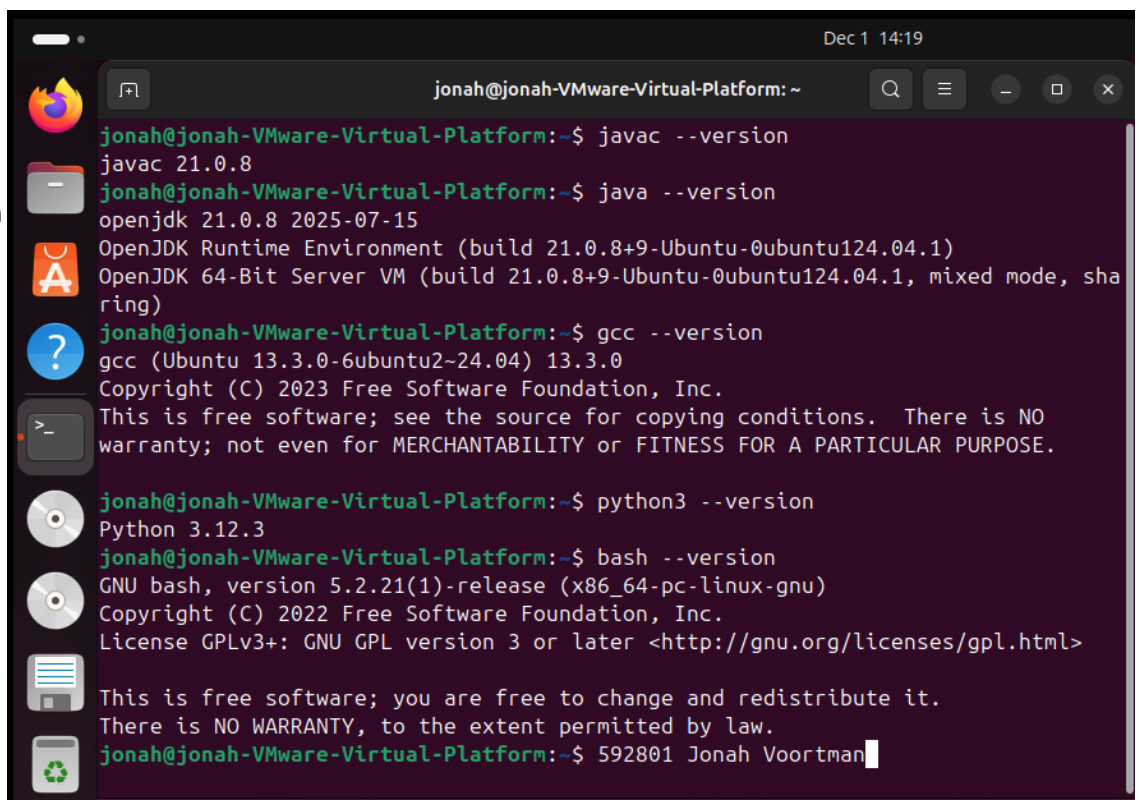
javac --version

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?

Fib.c en Fibonacci.java

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

Fib.c wordt fib_c, deze.

Which source code files are compiled to byte code?

Fibonacci.java naar Fibonacci.class

Which source code files are interpreted by an interpreter?

Fib.py en 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?

Bvb. Java Fibonacci.java

How do I run a Python program?

Bvb. python3 fib.py

How do I run a C program?

Bvb. ./fib_c

How do I run a Bash script?

Bvb. chmod +x fib.sh

./fib.sh

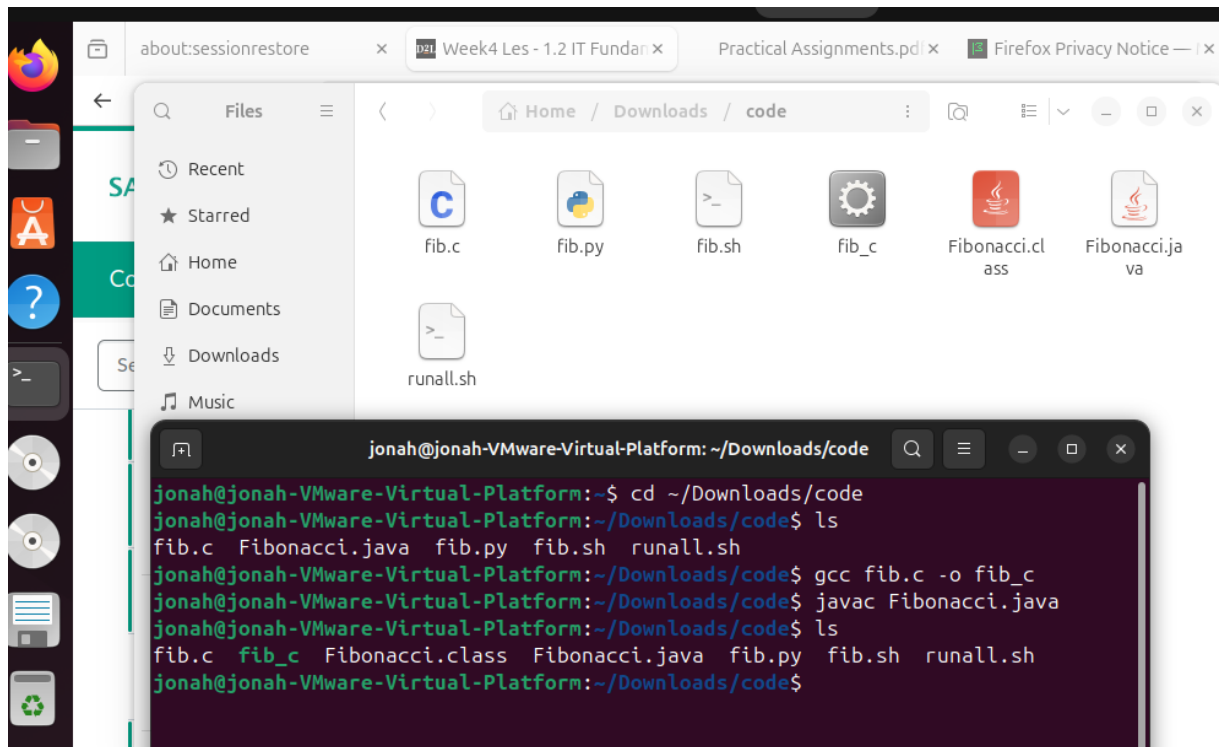
If I compile the above source code, will a new file be created? If so, which file?

Fib.c wordt fib_c en Fibonacci.java wordt Fibonacci.class

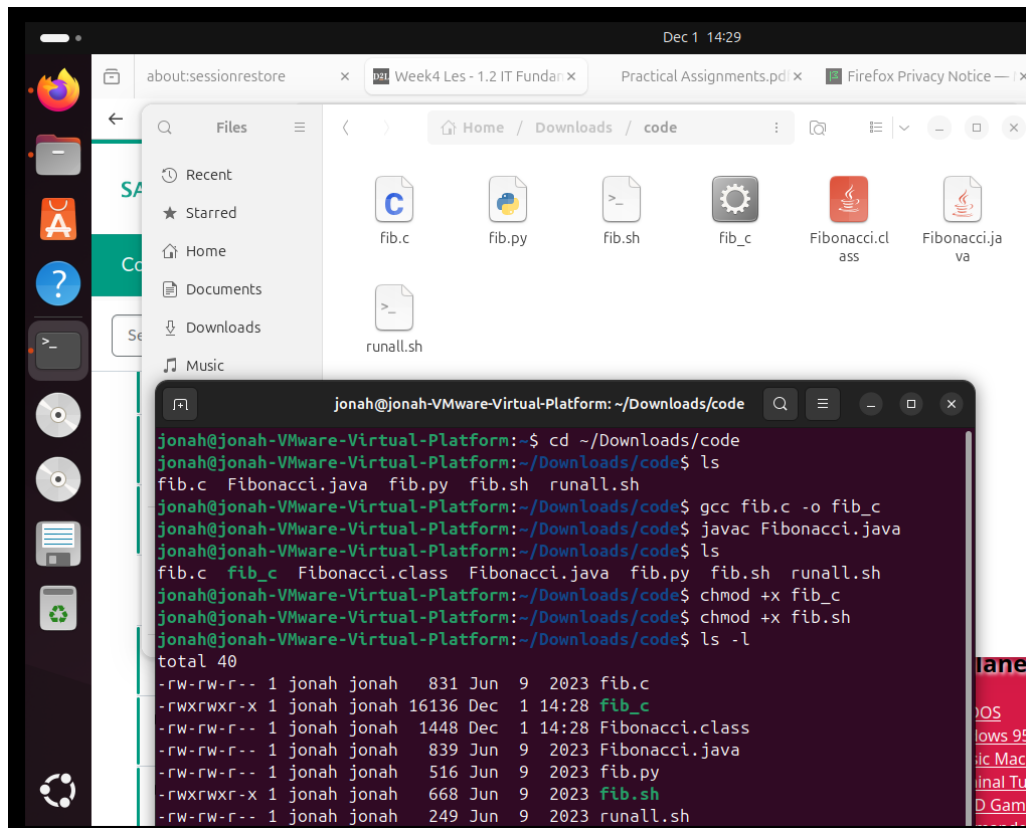
Take relevant screenshots of the following commands:

- Compile the source files where necessary
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?

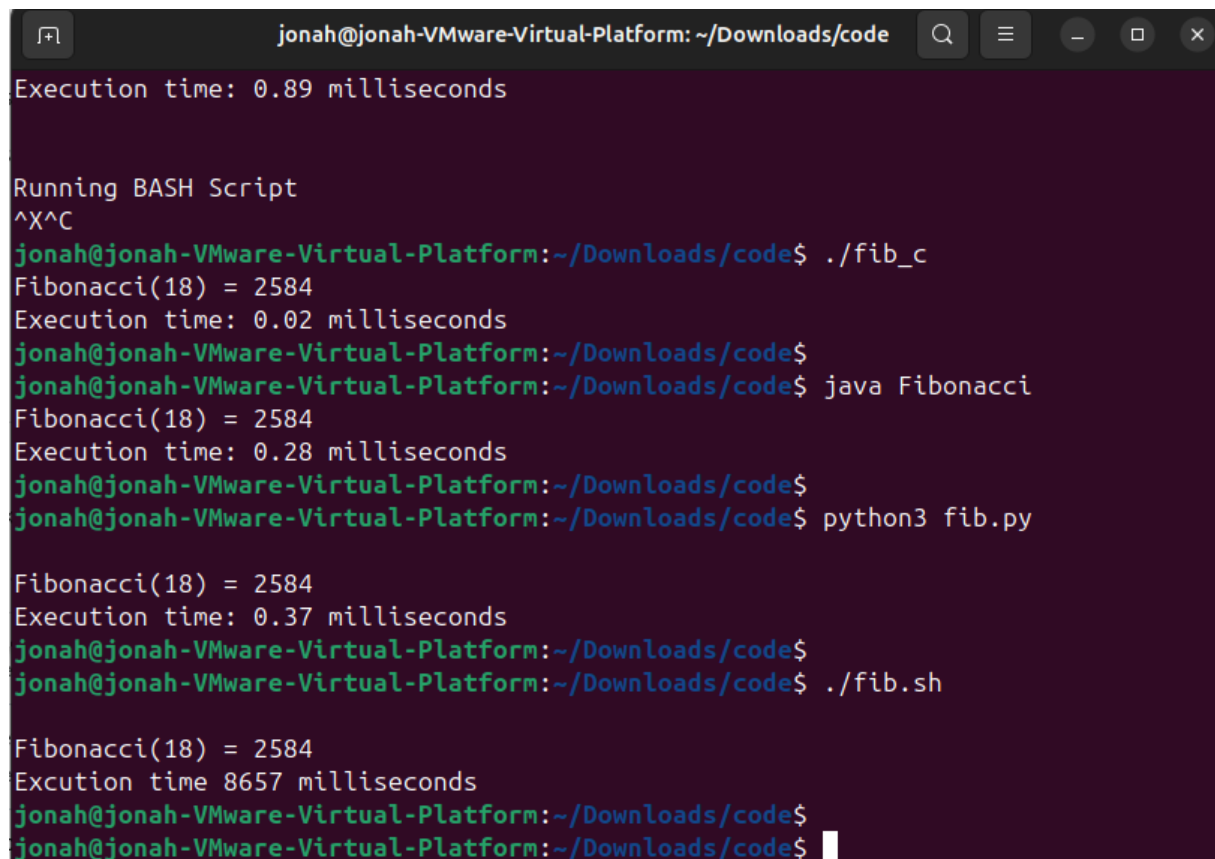
Compiling the files



Executable maken



Hier run ik ze, je ziet het C programma doet het het snelst



```
jonah@jonah-VMware-Virtual-Platform: ~/Downloads/code
Execution time: 0.89 milliseconds

Running BASH Script
^X^C
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ ./fib_c
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.28 milliseconds
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py

Fibonacci(18) = 2584
Execution time: 0.37 milliseconds
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ ./fib.sh

Fibonacci(18) = 2584
Execution time 8657 milliseconds
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$
```

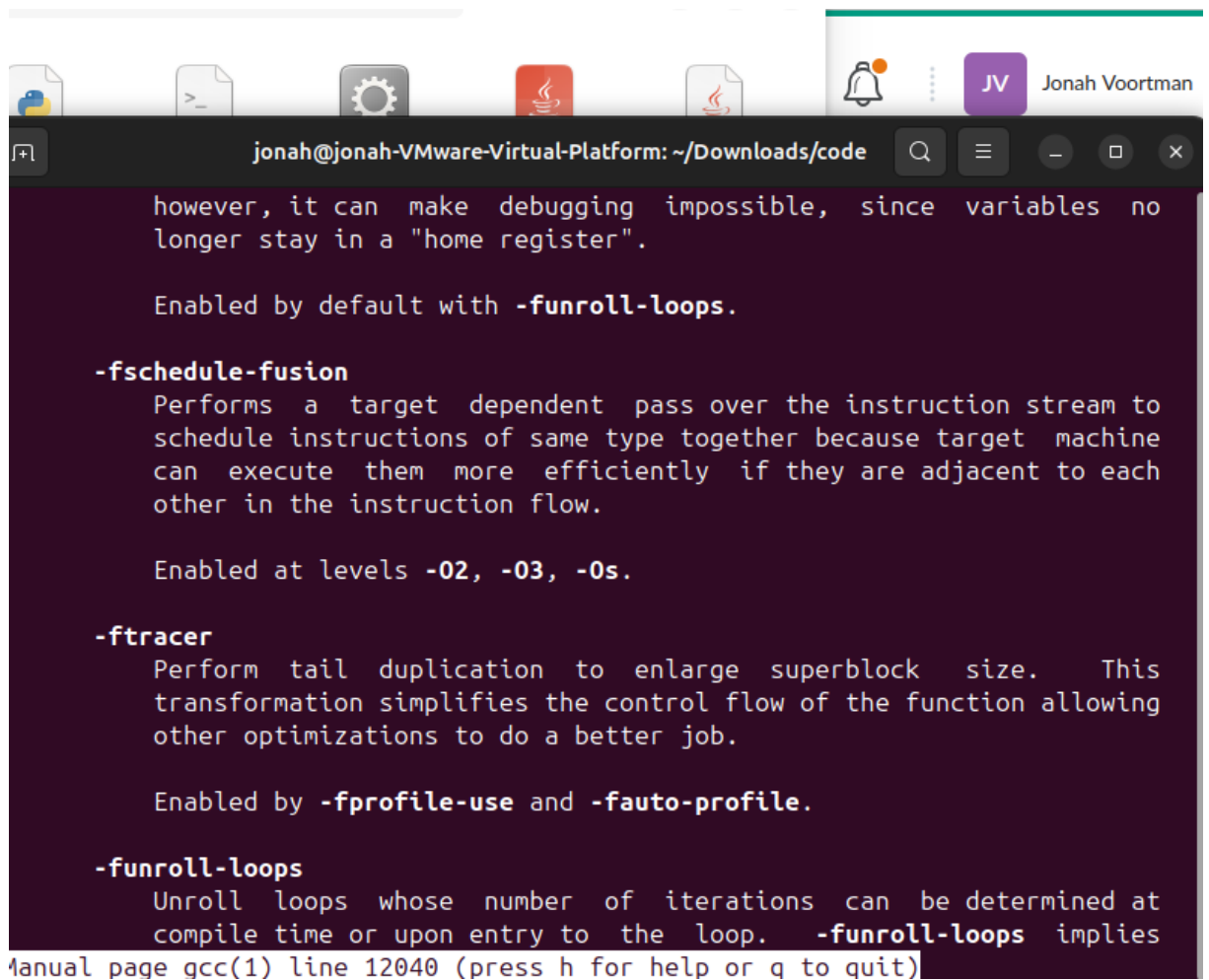
Assignment 4.4: Optimize

C word gecompileerd naar native machine code, dat draait direct op de cpu dus dat is het snelst

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.

Ik zie dat `-fschedule-fusion` geactiveerd is bij level `-O2`

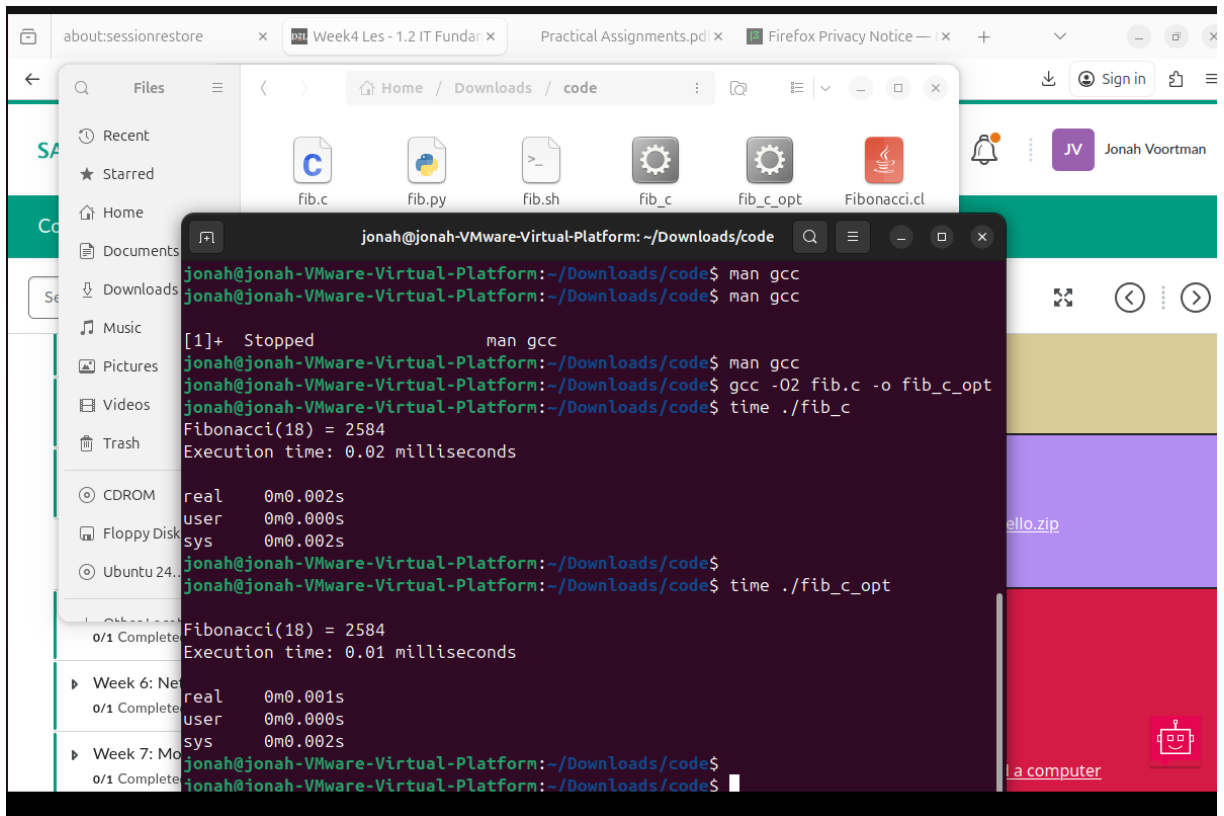


The screenshot shows a terminal window titled "jonah@jonah-VMware-Virtual-Platform: ~/Downloads/code". The terminal output describes several GCC optimization options:

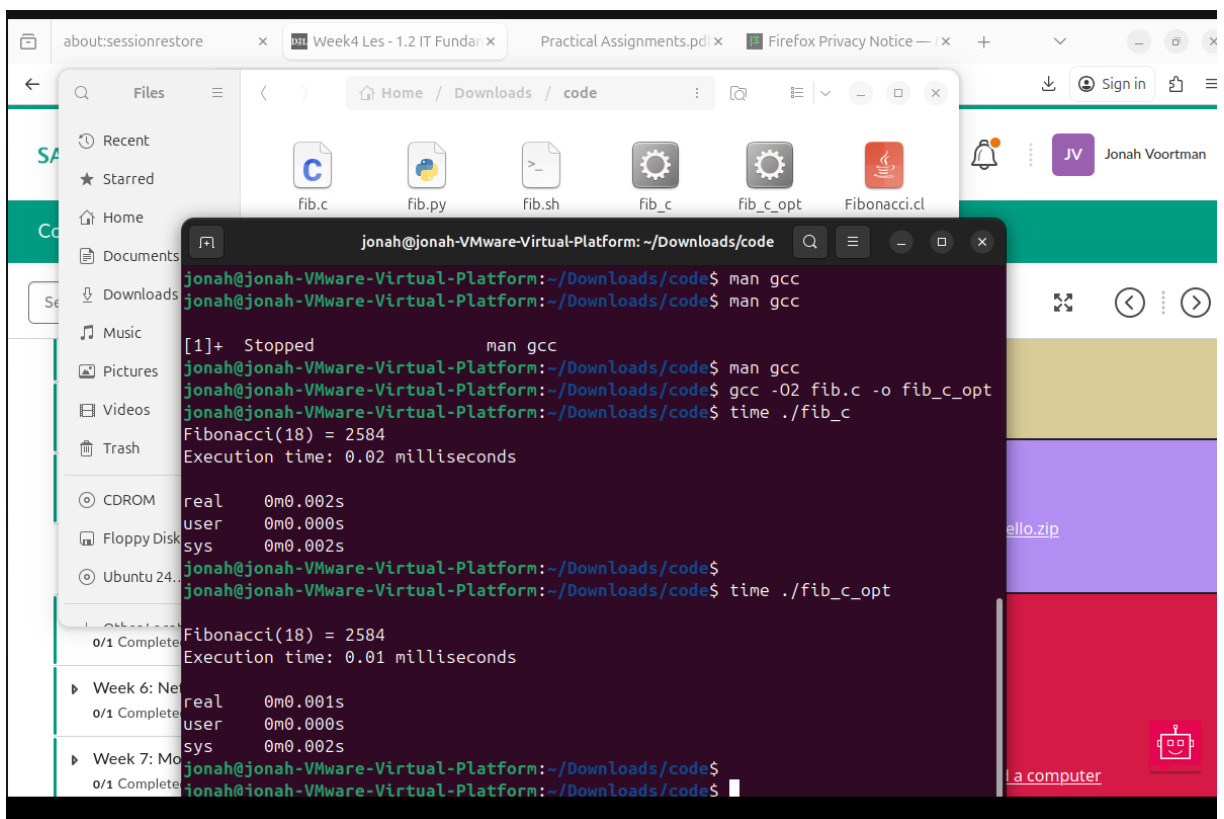
- `-funroll-loops`: Enabled by default with `-funroll-loops`.
- `-fschedule-fusion`: Performs a target dependent pass over the instruction stream to schedule instructions of same type together because target machine can execute them more efficiently if they are adjacent to each other in the instruction flow. Enabled at levels `-O2`, `-O3`, `-Os`.
- `-ftracer`: Perform tail duplication to enlarge superblock size. This transformation simplifies the control flow of the function allowing other optimizations to do a better job. Enabled by `-fprofile-use` and `-fauto-profile`.
- `-funroll-loops`: Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. `-funroll-loops` implies

Manual page gcc(1) line 12040 (press h for help or q to quit)

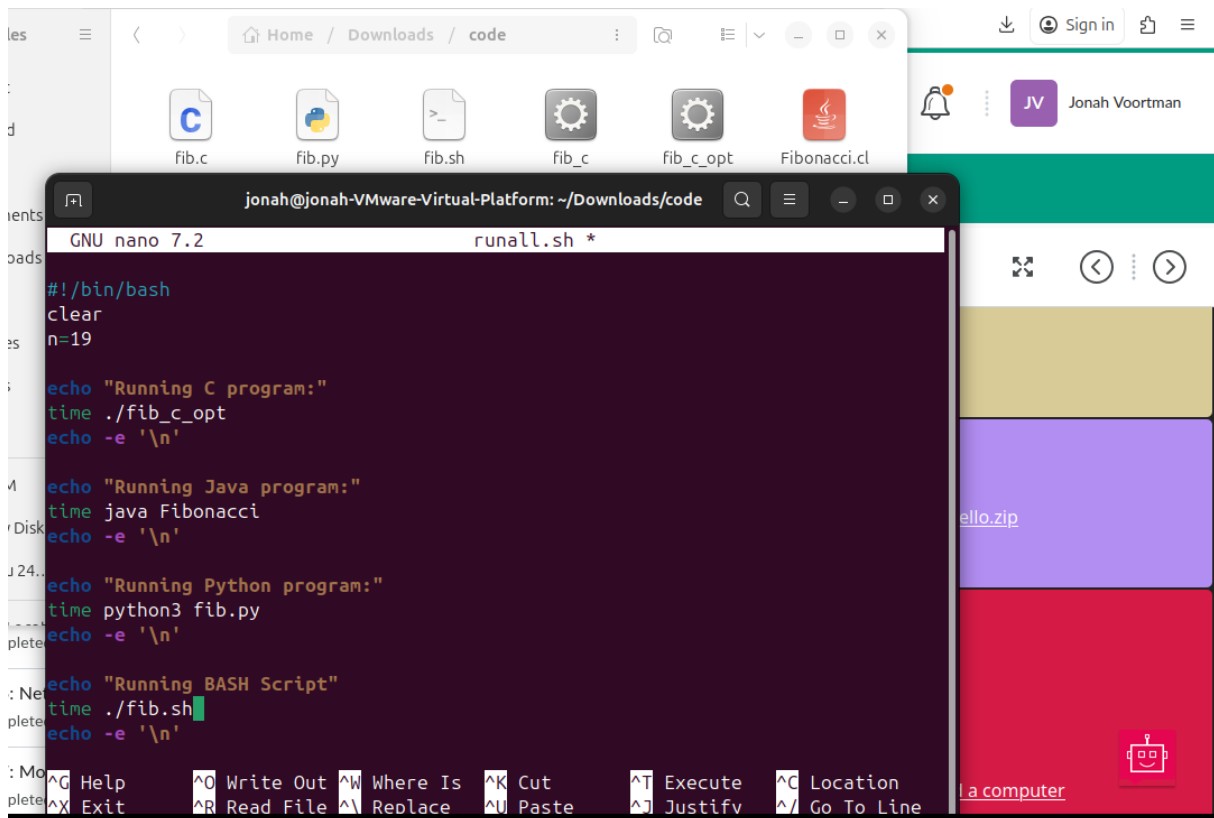
b) Compile `fib.c` again with the optimization parameters



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



- 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.



The screenshot shows a file manager window with the following files: fib.c, fib.py, fib.sh, fib_c, fib_c_opt, and Fibonacci.cl. Below the file manager is a terminal window titled 'jonah@jonah-VMware-Virtual-Platform: ~/Downloads/code'. The terminal is running GNU nano 7.2 and editing a file named 'runall.sh'. The script content is as follows:

```
#!/bin/bash
clear
n=19

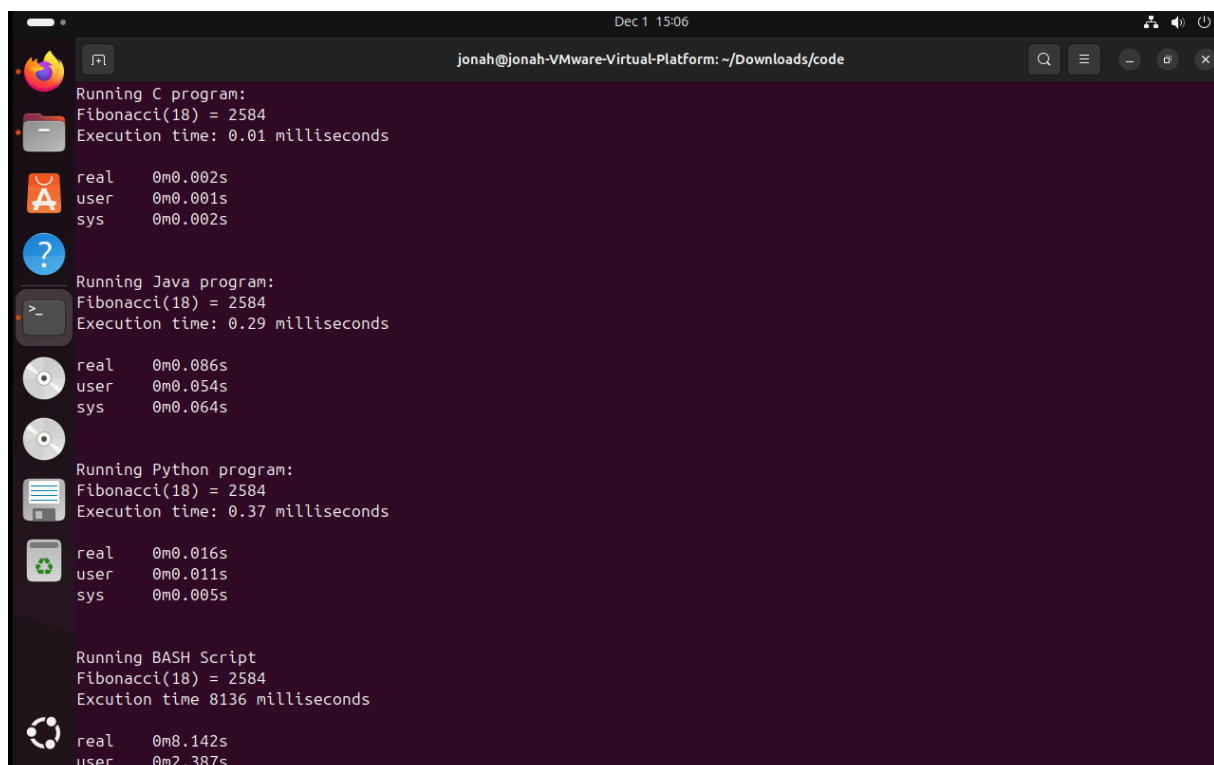
echo "Running C program:"
time ./fib_c_opt
echo -e '\n'

echo "Running Java program:"
time java Fibonacci
echo -e '\n'

echo "Running Python program:"
time python3 fib.py
echo -e '\n'

echo "Running BASH Script"
time ./fib.sh
echo -e '\n'
```

The terminal window also shows a status bar with various keyboard shortcuts like ^G Help, ^O Write Out, ^W Where Is, ^K Cut, ^T Execute, ^C Location, ^X Exit, ^R Read File, ^A Replace, ^U Paste, ^J Justify, and ^_ Go To Line.



The screenshot shows the terminal window after executing the 'runall.sh' script. The output is as follows:

```
Running C program:
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.001s
sys     0m0.002s

Running Java program:
Fibonacci(18) = 2584
Execution time: 0.29 milliseconds

real    0m0.086s
user    0m0.054s
sys     0m0.064s

Running Python program:
Fibonacci(18) = 2584
Execution time: 0.37 milliseconds

real    0m0.016s
user    0m0.011s
sys     0m0.005s

Running BASH Script
Fibonacci(18) = 2584
Execution time 8136 milliseconds

real    0m8.142s
user    0m2.387s
```

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
```

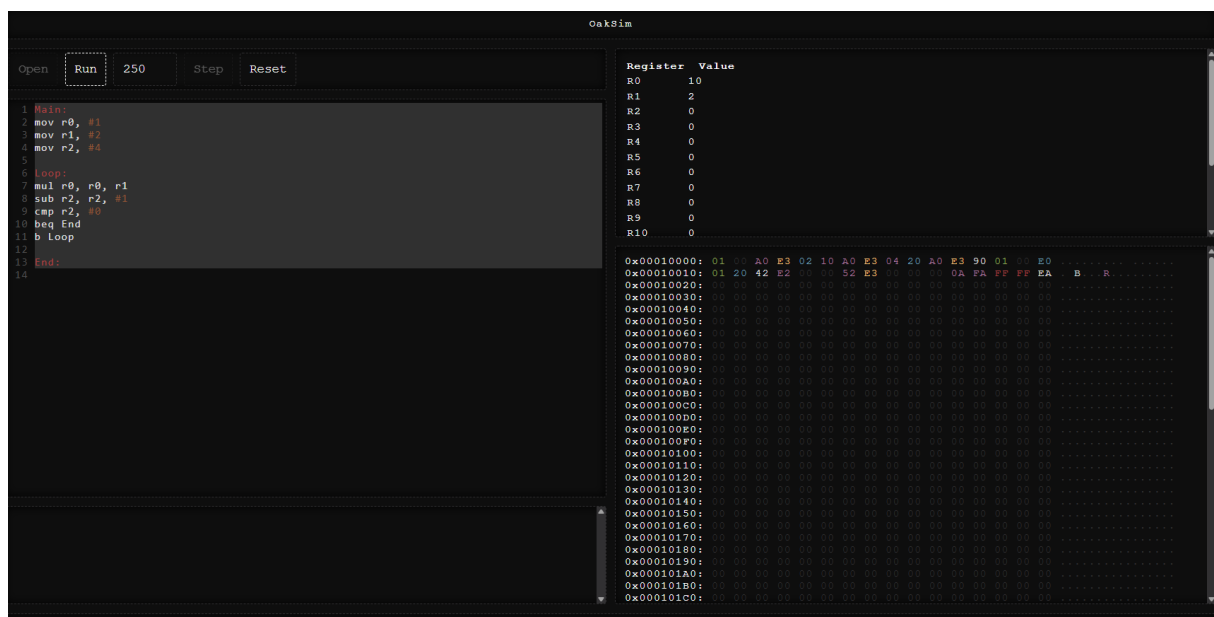
```
mov r2, #4
```

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](#)