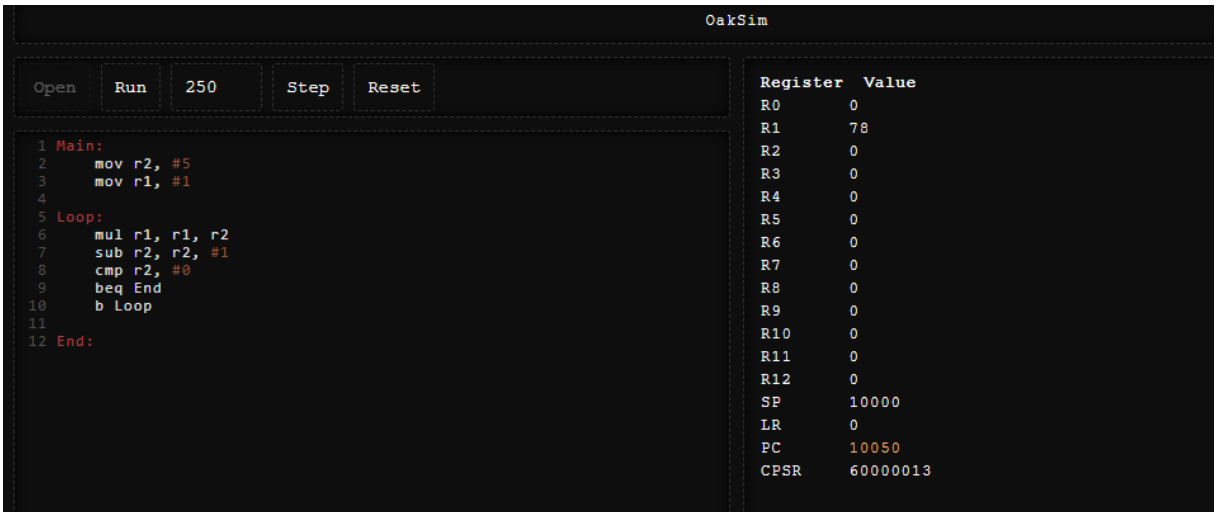
Template Week 4 – Software

Student number: 591007

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
    mov r2, #5
    mov r1, #1

Loop:
    mul r1, r1, r2
    sub r2, r2, #1
    cmp r2, #0
    beq End
    b Loop

End:
```

```
Register  Value
R0        0
R1        78
R2        0
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
R11       0
R12       0
SP        10000
LR        0
PC        10050
CPSR      60000013
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

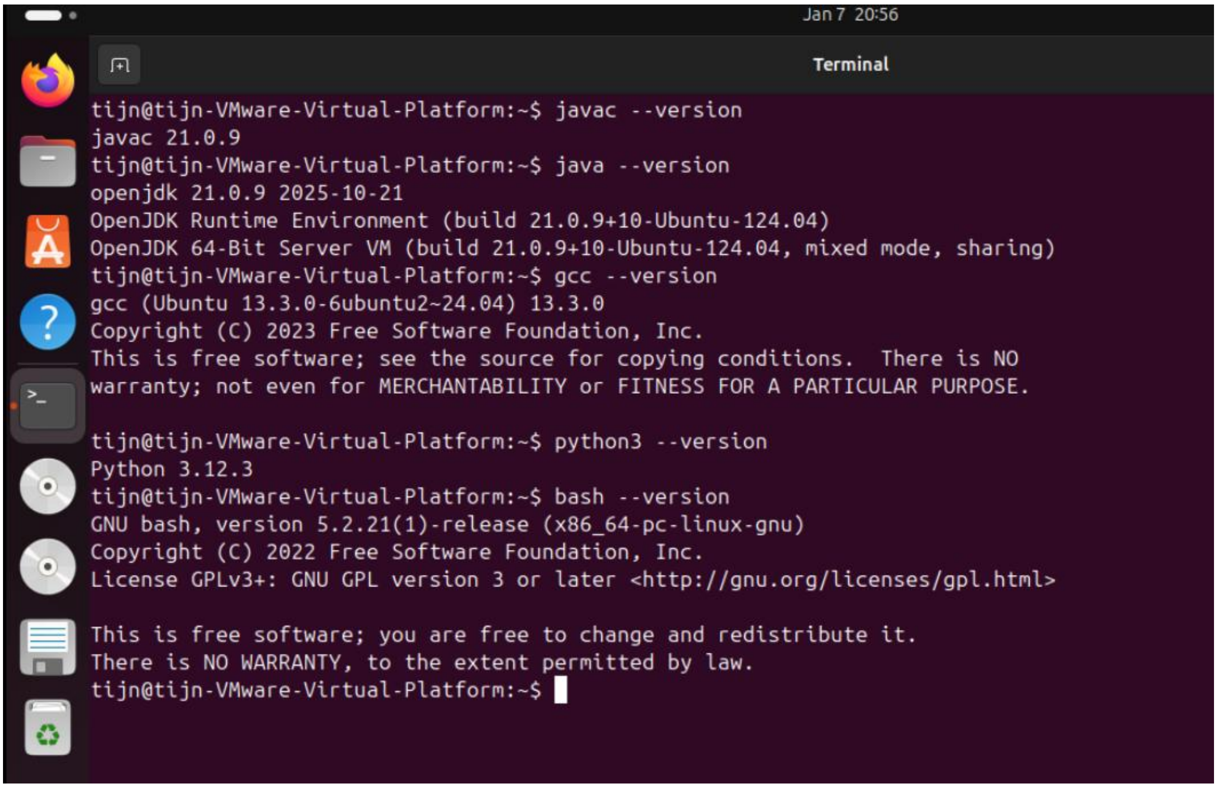
javac --version

java --version

gcc --version

python3 --version

bash --version



```
tijn@tijn-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
tijn@tijn-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)
tijn@tijn-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.

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

Assignment 4.3: Compile

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

Fibonacci.java moet gecompileerd worden via javac

Fib.c moet gecompileerd worden met gcc

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

Fib.c word gecompileerd naar machine code

Which source code files are compiled to byte code?

Fibonacci.java word gecompileerd naar java bytecode

Which source code files are interpreted by an interpreter?

Fib.py word geinterperd door de python interpeter.

Fib.sh word geinterperd door de bash shell.

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

Fib.c want deze word meteen gecompileerd naar machine code.

How do I run a Java program?

Eerst moet ik het compileren via: **javac Fibonacci.java** en daarna kan ik via: **java Fibonacci** het programma runnen.

How do I run a Python program?

Dit kan ik direct runnen via: **python3 fib.py**

How do I run a C program?

Eerst zou ik deze moeten compileren via: **gcc fib.c -0 fib** en dan kan ik de executable runnen via: **./fib**

How do I run a Bash script?

Eerst moet ik hier een executable van maken via: **sudo chmod a+x fib.sh** deze kan ik daarna runnen via: **sudo ./fib.sh**

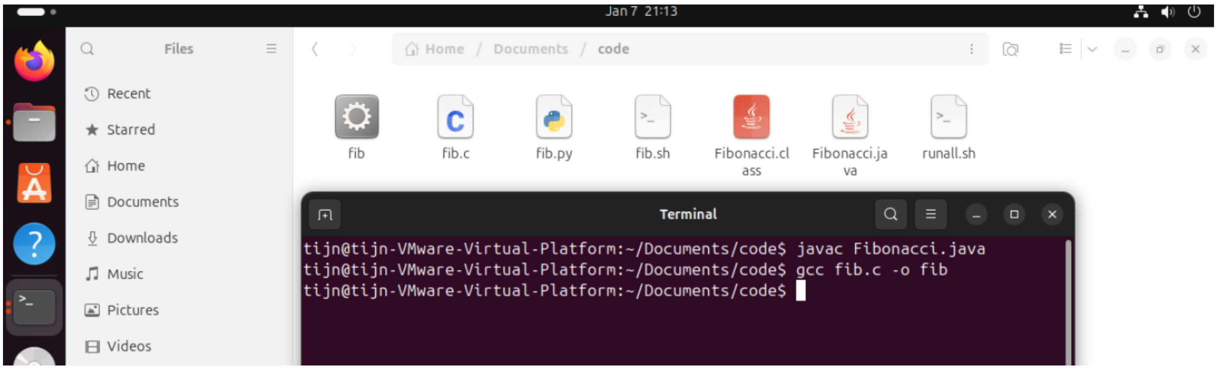
If I compile the above source code, will a new file be created? If so, which file?

Ja java maakt een klasse aan genaamd **Fibonacci.class**

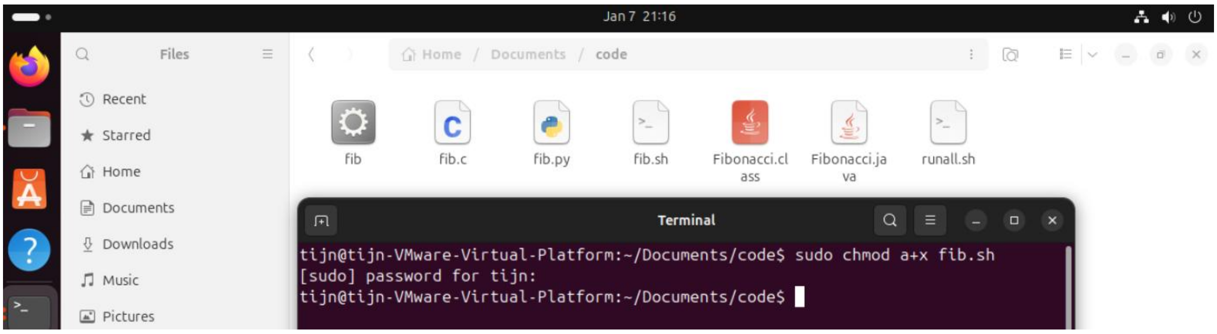
C code maakt ook een nieuwe file aan genaamd fib

Take relevant screenshots of the following commands:

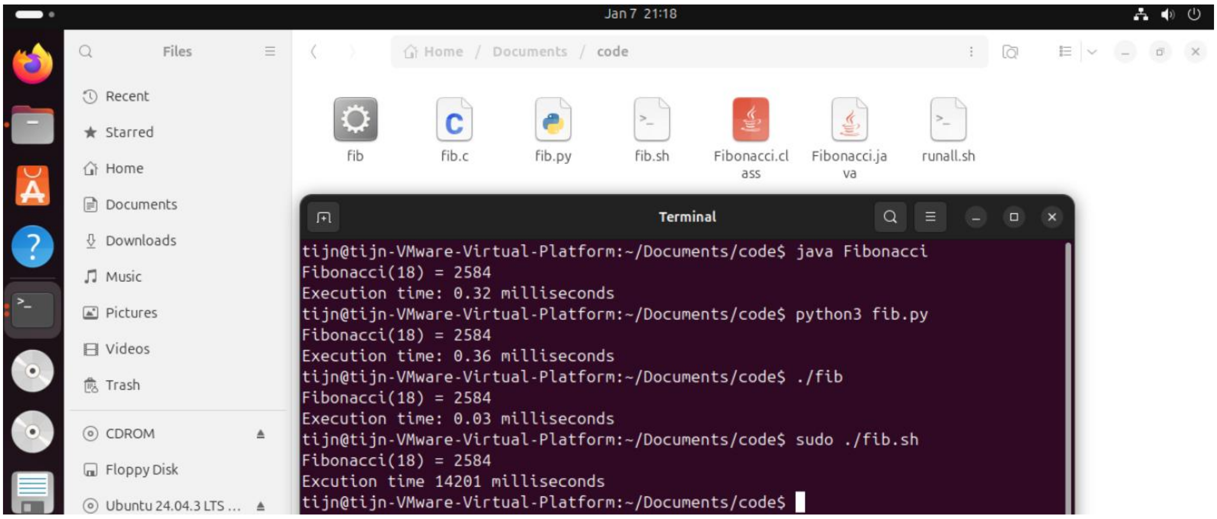
- Compile the source files where necessary
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?



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



```
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ sudo chmod a+x fib.sh
[sudo] password for tijn:
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$
```



```
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.32 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.36 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 14201 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$
```

Zoals verwacht is de gecompileerde C code het snelst.

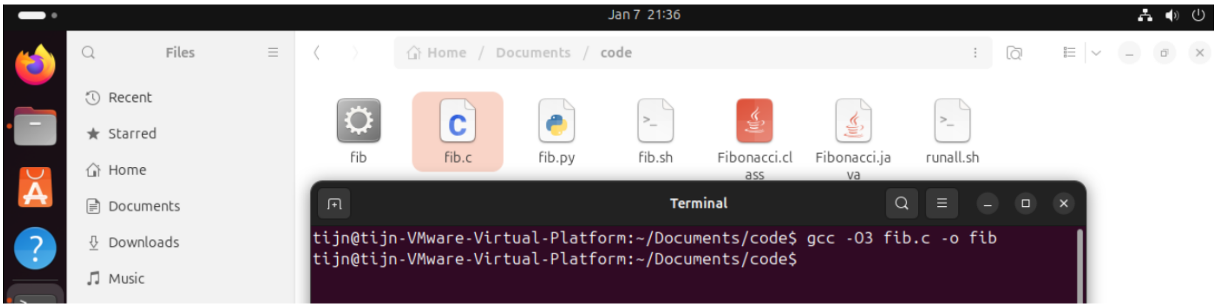
Assignment 4.4: Optimize

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.

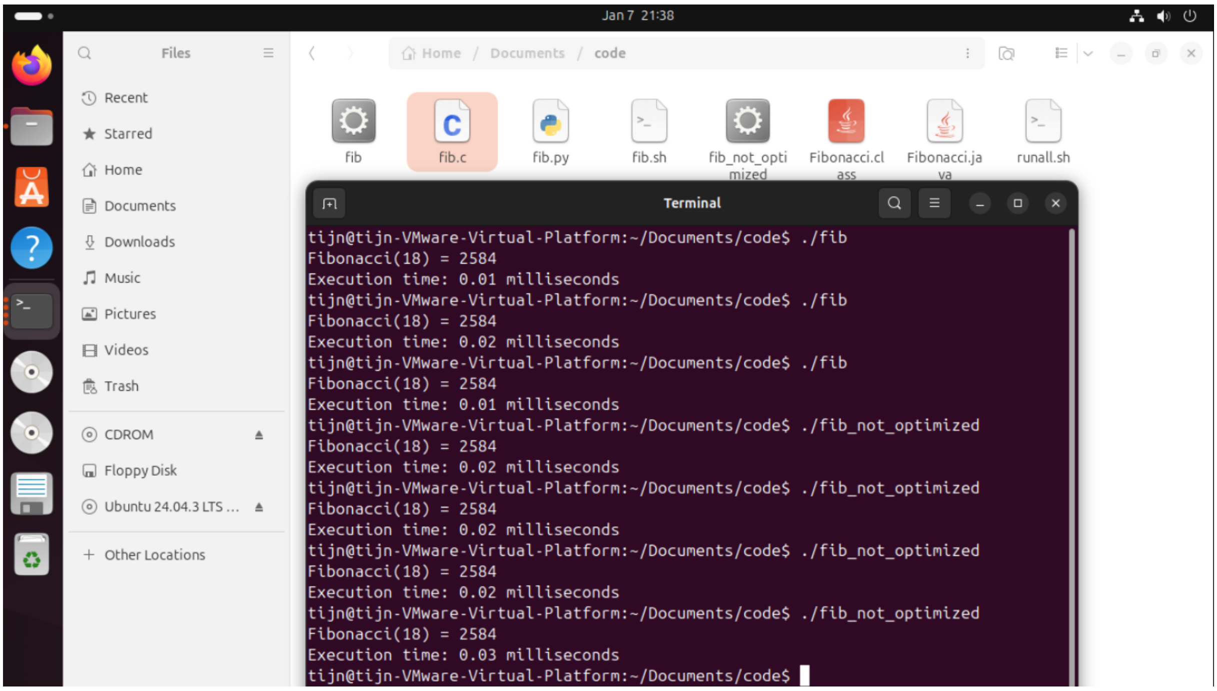
Ik ga parameter -O3 meegeven. Dit ga ik dan doen doormiddel van: **gcc -O3 fib.c -o fib**

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



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

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

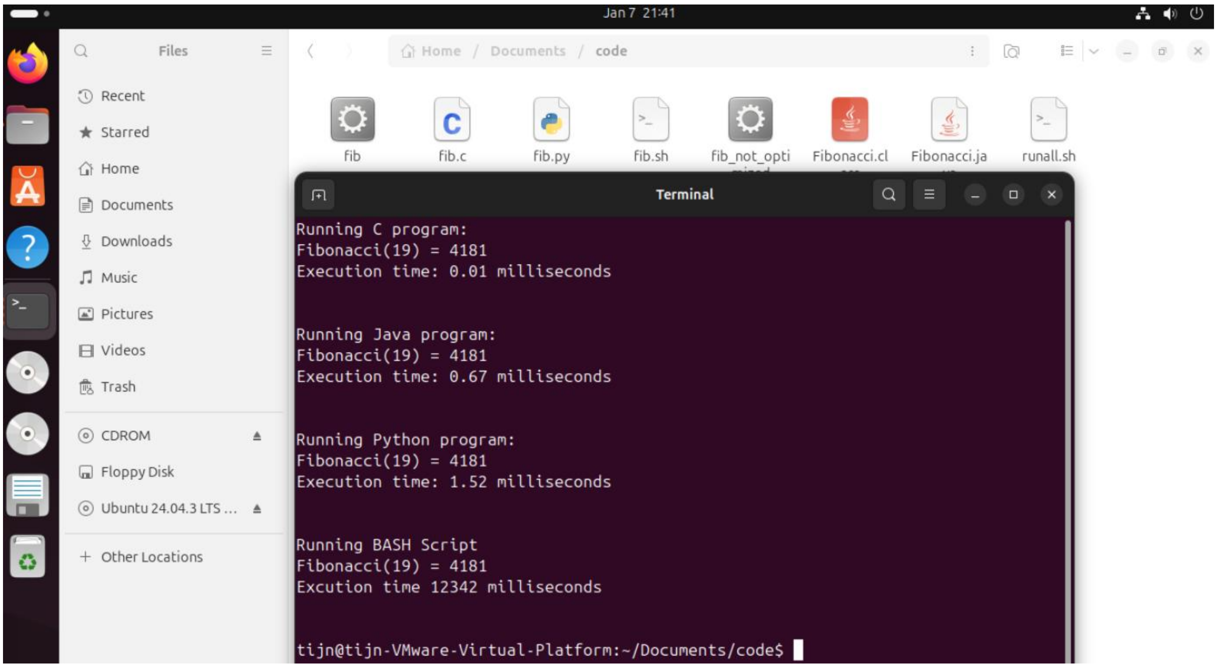


```
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib_not_optimized
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib_not_optimized
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib_not_optimized
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$ ./fib_not_optimized
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
tijn@tijn-VMware-Virtual-Platform:~/Documents/code$
```

Het runt gemiddeld 1 milliseconde sneller dan het niet geoptimaliseerde programma.

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.

De runall.sh werkte al dus hoefde hier niks aan aanpassen.



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


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.67 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 1.52 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 12342 milliseconds


tijn@tijn-VMware-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.

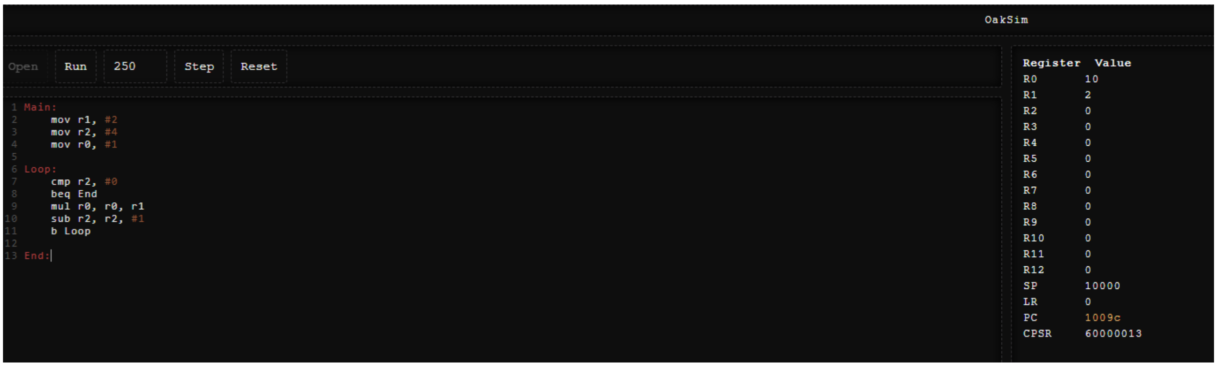
```
Main:
     mov r1, #2
     mov r2, #4
     mov r0, #1

Loop:
     cmp r2, #0
     beq End
     mul r0, r0, r1
     sub r2, r2, #1
     b Loop

End:
```

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

Screenshot of the completed code here.



```
Main:
    mov r1, #2
    mov r2, #4
    mov r0, #1

Loop:
    cmp r2, #0
    beq End
    mul r0, r0, r1
    sub r2, r2, #1
    b Loop

End:
```

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
LR	0
PC	1009c
CPSR	60000013

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