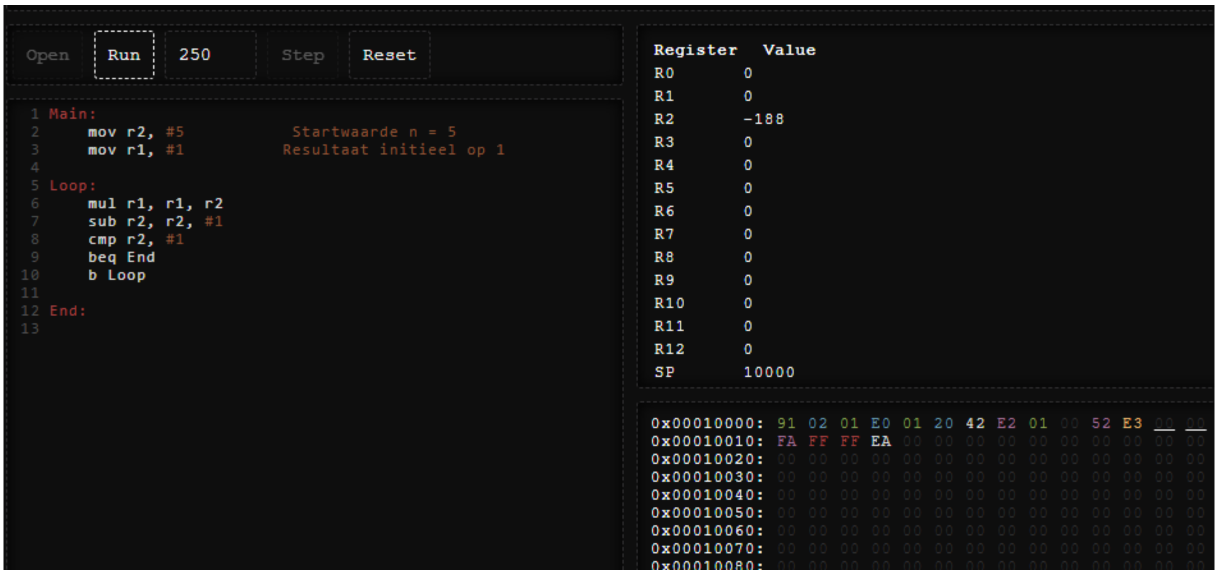
Template Week 4 – Software

Student number: 593250

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
    mov r2, #5          Startwaarde n = 5
    mov r1, #1         Resultaat initieel op 1

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

End:
```

```
Register  Value
R0        0
R1        0
R2        -188
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
R11       0
R12       0
SP        10000
```

```
0x00010000: 91 02 01 E0 01 20 42 E2 01 00 52 E3 00 00
0x00010010: FA FF FF EA 00 00 00 00 00 00 00 00 00 00
0x00010020: 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
0x00010040: 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
0x00010060: 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
0x00010080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

1. r2 telt af van 5 naar 1

2. r1 bouwt de factorialwaarde op

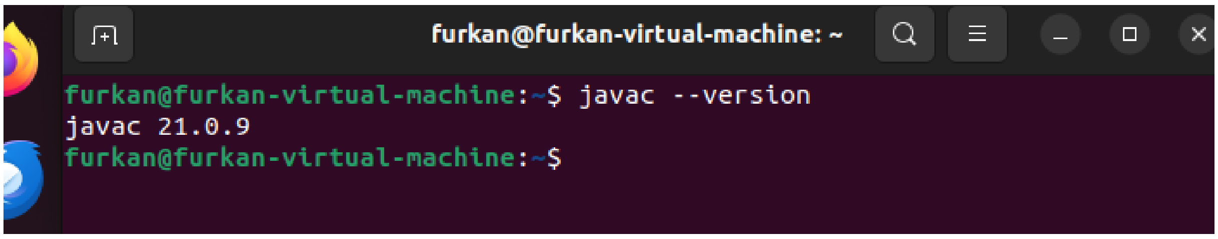
3. Zodra r2 1 bereikt, stopt de loop

4. Het resultaat van 5! = 120 staat dan in r1

Assignment 4.2: Programming languages

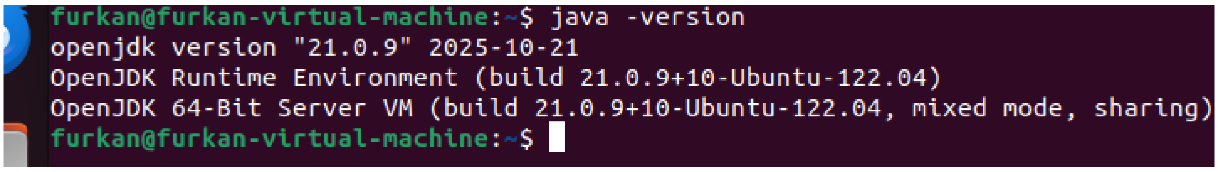
Take screenshots that the following commands work:

javac –version



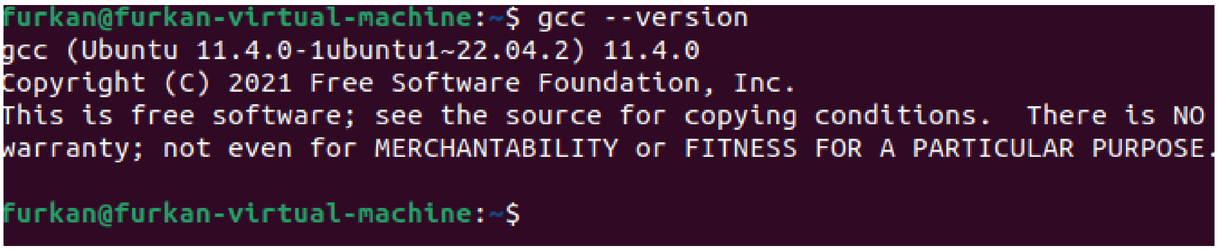
```
furkan@furkan-virtual-machine: ~
furkan@furkan-virtual-machine:~$ javac --version
javac 21.0.9
furkan@furkan-virtual-machine:~$
```

java –version



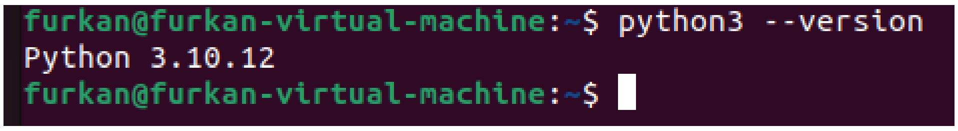
```
furkan@furkan-virtual-machine:~$ java -version
openjdk version "21.0.9" 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-122.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-122.04, mixed mode, sharing)
furkan@furkan-virtual-machine:~$
```

gcc –version



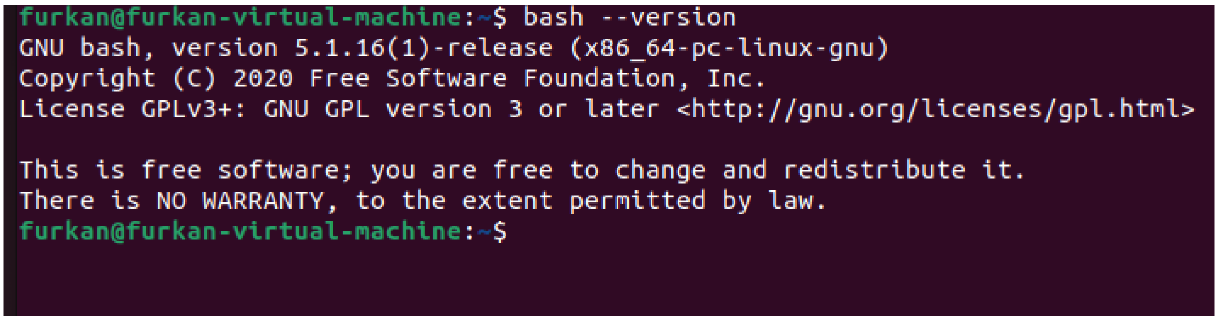
```
furkan@furkan-virtual-machine:~$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

furkan@furkan-virtual-machine:~$
```

python3 –version



```
furkan@furkan-virtual-machine:~$ python3 --version
Python 3.10.12
furkan@furkan-virtual-machine:~$
```

bash –version



```
furkan@furkan-virtual-machine:~$ bash --version
GNU bash, version 5.1.16(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2020 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.
furkan@furkan-virtual-machine:~$
```

Assignment 4.3: Compile

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

1. **Fibonacci.java**: moet gecompileerd worden naar bytecode (.class bestand)
2. **fib.c**: moet gecompileerd worden naar machinecode (een executable)
3. **fib.py**: hoeft niet gecompileerd te worden, wordt geïnterpreteerd door Python interpreter
4. **fib.sh**: bash script, wordt uitgevoerd door de shell, niet gecompileerd

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

fib.c (C broncode wordt gecompileerd naar native machinecode, een uitvoerbaar bestand)

Which source code files are compiled to byte code?

Fibonacci.java (Java broncode wordt gecompileerd naar Java bytecode, uitgevoerd door de JVM)

Which source code files are interpreted by an interpreter?

fib.py (Python script wordt geïnterpreteerd door de Python interpreter)

fib.sh (Bash script wordt geïnterpreteerd door de 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 is het snelst (native machinecode)
Daarna Java, daarna Python, en als laatste Bash script.

How do I run a Java program?

```
javac Fibonacci.java    # compileer Java broncode

java Fibonacci          # start de JVM en voer uit
```

How do I run a Python program?

```
python3 fib.py
```

How do I run a C program?

```
gcc fib.c -o fib     # compileer C broncode naar executable

./fib                # voer het programma uit
```

How do I run a Bash script?

sudo chmod a+x fib.sh   # maak het script uitvoerbaar (1x nodig)

./fib.sh   # voer het script uit

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

Java: er wordt een Fibonacci.class bestand gemaakt (bytecode)

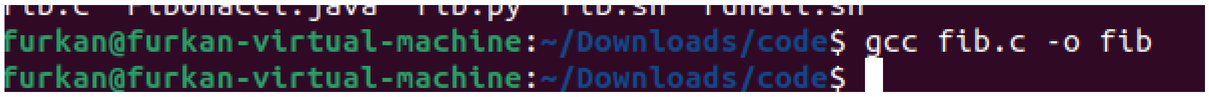
C: er wordt een uitvoerbaar bestand gemaakt, meestal fib (of wat je opgeeft met -o)

Python en Bash: geen nieuw bestand, ze worden geïnterpreteerd

Take relevant screenshots of the following commands:

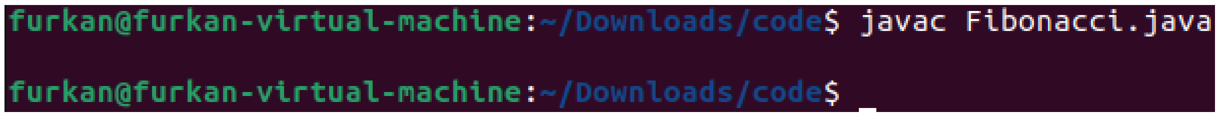
- Compile the source files where necessary

  C compile



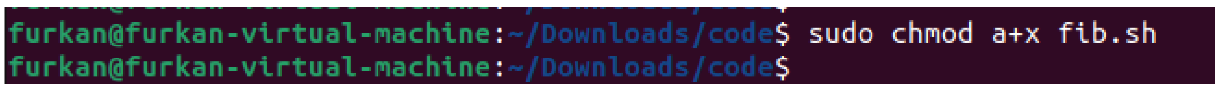
```
fib.c   Fibonacci.java   fib.py   fib.sh   runall.sh
furkan@furkan-virtual-machine:~/Downloads/code$ gcc fib.c -o fib
furkan@furkan-virtual-machine:~/Downloads/code$
```

  Java Compile



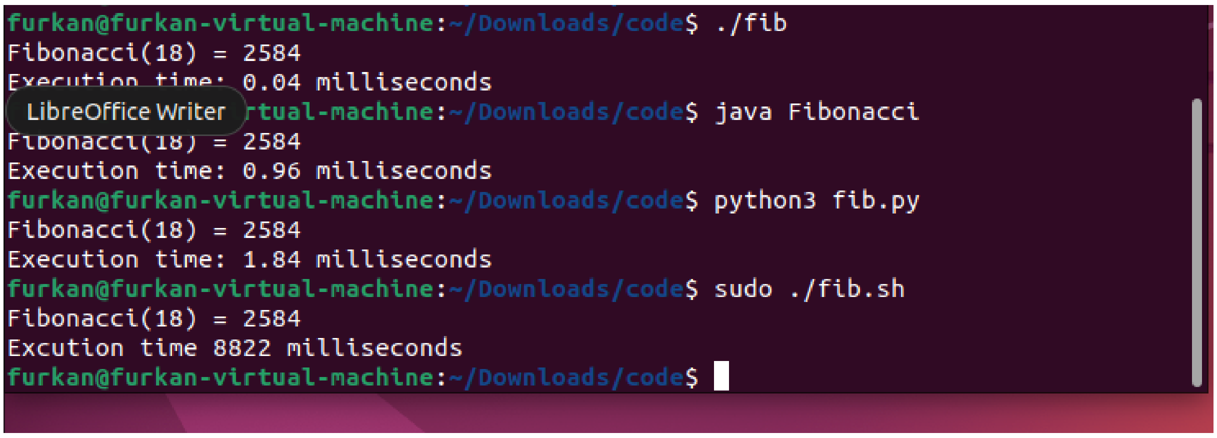
```
furkan@furkan-virtual-machine:~/Downloads/code$ javac Fibonacci.java

furkan@furkan-virtual-machine:~/Downloads/code$
```

- Make them executable



```
furkan@furkan-virtual-machine:~/Downloads/code$ sudo chmod a+x fib.sh
furkan@furkan-virtual-machine:~/Downloads/code$
```

- Run them



```
furkan@furkan-virtual-machine:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.04 milliseconds
furkan@furkan-virtual-machine:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.96 milliseconds
furkan@furkan-virtual-machine:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 1.84 milliseconds
furkan@furkan-virtual-machine:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 8822 milliseconds
furkan@furkan-virtual-machine:~/Downloads/code$
```

- Which (compiled) source code file performs the calculation the fastest?
  Het snelste programma is het C-programma (fib).

Dit komt doordat C-code wordt **gecompileerd naar directe machinecode**, waardoor de CPU het programma zonder extra tussenlagen kan uitvoeren.
Java is iets langzamer omdat het in bytecode draait binnen de JVM.
Python en Bash zijn geïnterpreteerde talen en zijn daardoor het traagst.

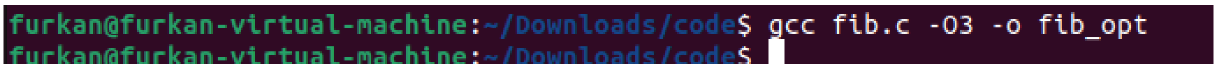
Assignment 4.4: Optimize

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.

De **gcc-optimalisatieparameters** zijn:
-O0 geen optimalisatie
-O1 basisoptimalisaties
-O2 sterke optimalisaties
-O3 maximale optimalisatie (meestal de beste voor snelheid)
-Ofast nog agressiever, maar minder veilig en niet volledig standaard-compliant

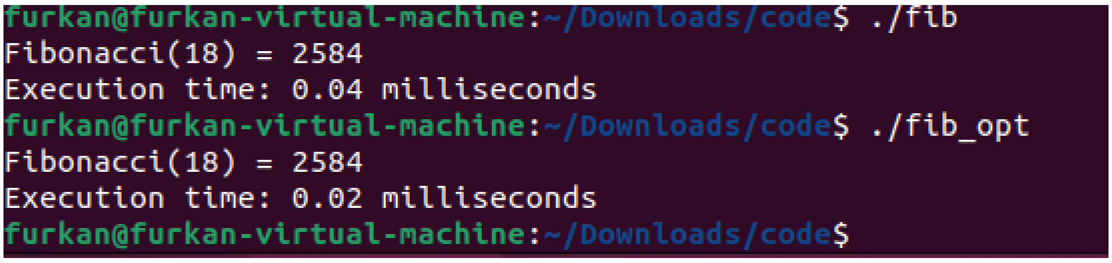
b) Compile **fib.c** again with the optimization parameters



```
furkan@furkan-virtual-machine:~/Downloads/code$ gcc fib.c -O3 -o fib_opt
furkan@furkan-virtual-machine:~/Downloads/code$
```

-O3 zorgt voor maximale compiler-optimalisaties, zoals loop unrolling, inline functies en agressieve snelheidsverbeteringen.

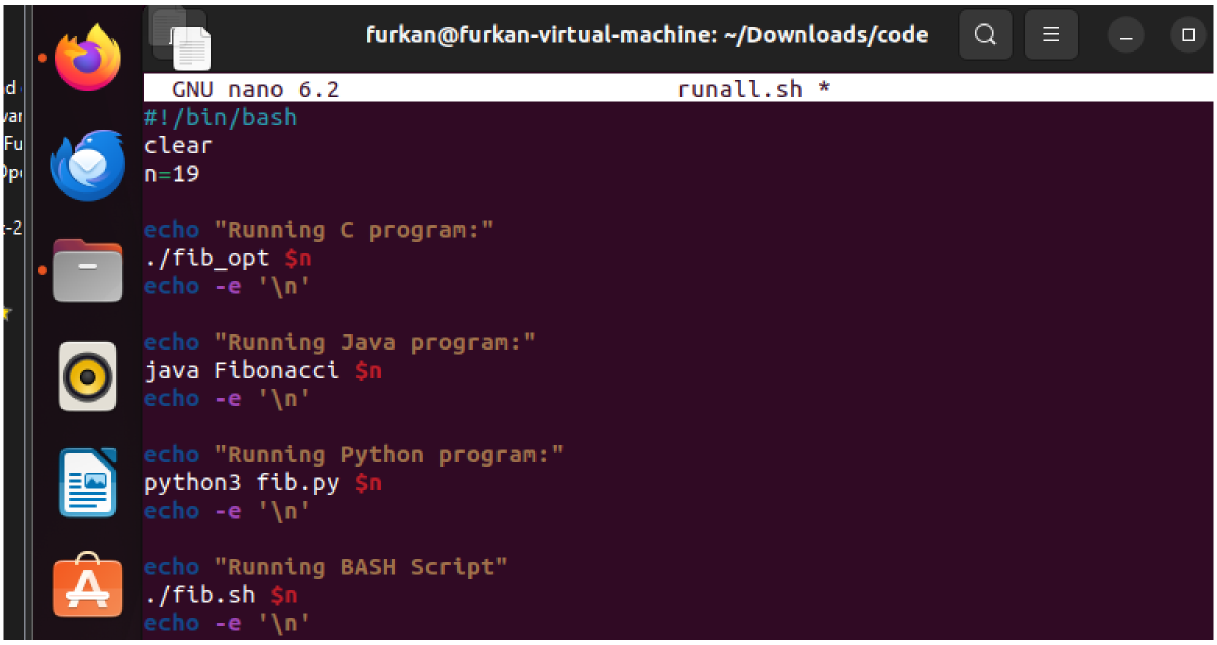
c) Run the newly compiled program. Is it true that it now performs the calculation faster?



```
furkan@furkan-virtual-machine:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.04 milliseconds
furkan@furkan-virtual-machine:~/Downloads/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
furkan@furkan-virtual-machine:~/Downloads/code$
```

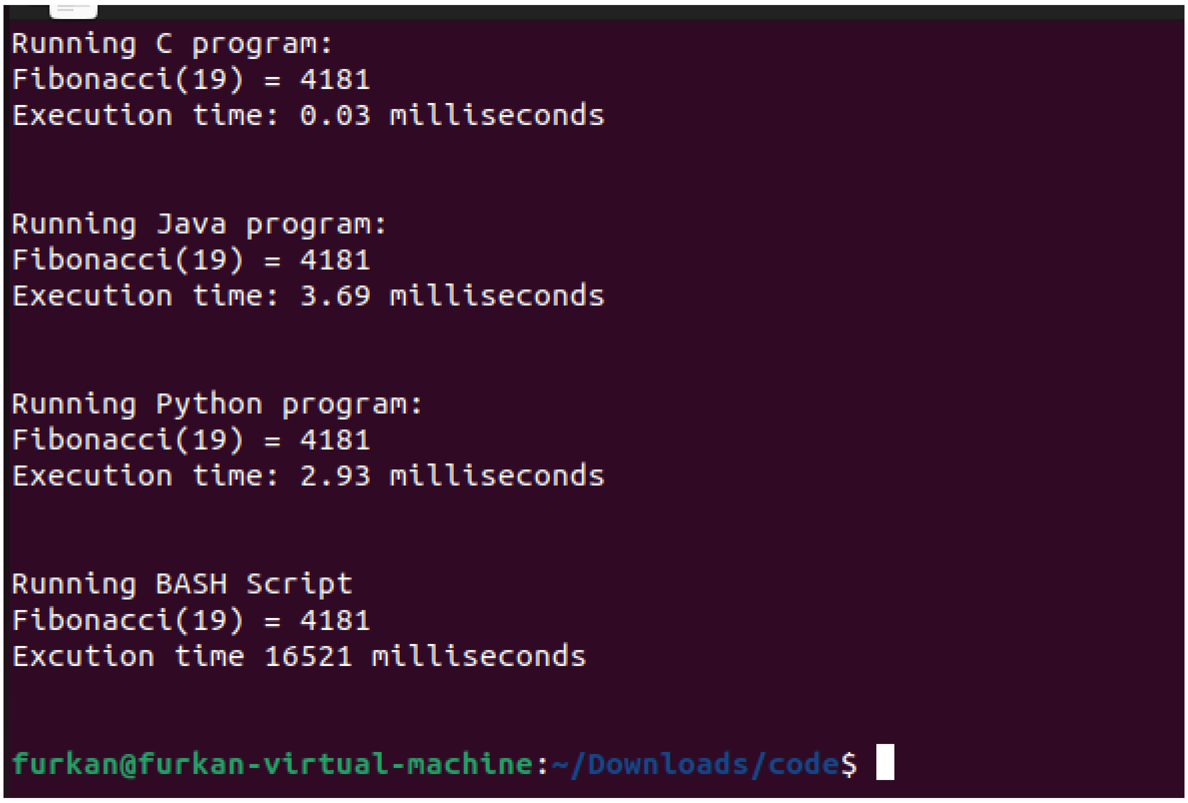
Ja de geoptimaliseerde versie **fib_opt** is sneller, omdat de compilter met -03 verbeteringen toepast zoals snellere CPU-Optimalisatie

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.



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

echo "Running C program:"
./fib_opt $n
echo -e '\n'

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

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

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

Bij de C script heb ik het aangepast naar ./fib_opt



```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.03 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 3.69 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 2.93 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 16521 milliseconds


furkan@furkan-virtual-machine:~/Downloads/code$
```

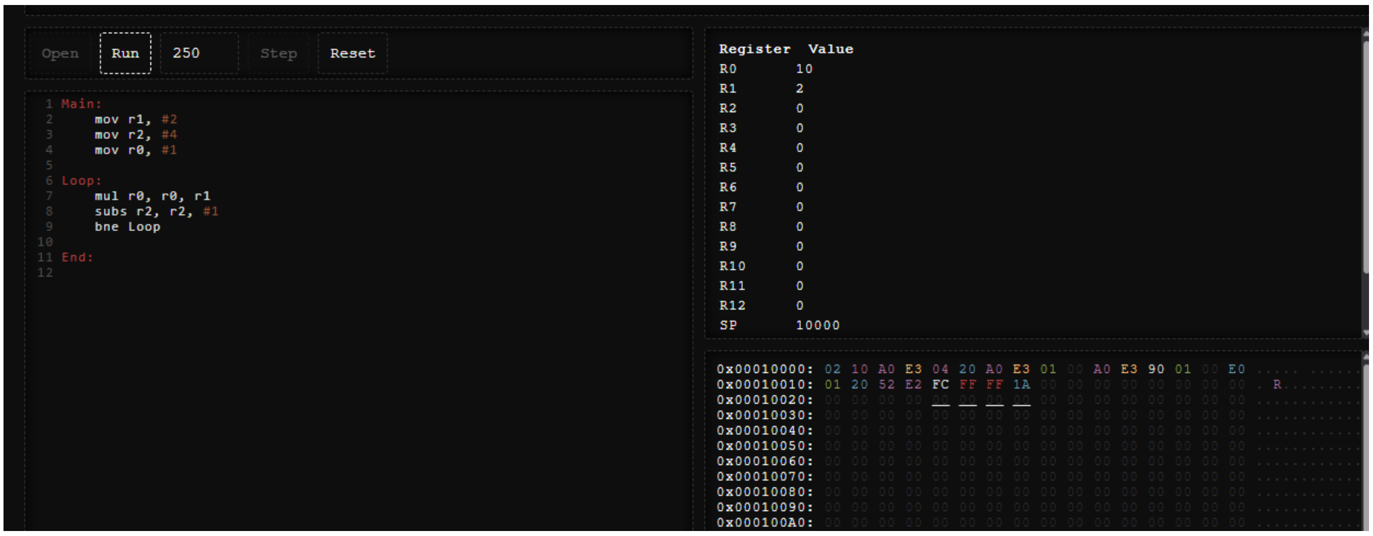
Assignment 4.5: More ARM Assembly

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


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:
    mul r0, r0, r1
    subs r2, r2, #1
    bne 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
```

```
0x00010000: 02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 90 01 00 E0 ................
0x00010010: 01 20 52 E2 FC FF FF 1A 00 00 00 00 00 00 00 00 . R.............
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 ................
0x000100A0: 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**