

# Template Week 4 – Software

Student number: 583293

## 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

```
finn@finn-VMware-Virtual-Platform:~$ javac --version
Command 'javac' not found, but can be installed with:
sudo apt install openjdk-17-jdk-headless # version 17.0.17+10-1~24.04, or
sudo apt install openjdk-21-jdk-headless # version 21.0.9+10-1~24.04
sudo apt install default-jdk             # version 2:1.17-75
sudo apt install ecj                     # version 3.32.0+eclipse4.26-2
sudo apt install openjdk-19-jdk-headless # version 19.0.2+7-4
sudo apt install openjdk-20-jdk-headless # version 20.0.2+9-1
sudo apt install openjdk-22-jdk-headless # version 22~22ea-1
sudo apt install openjdk-11-jdk-headless # version 11.0.29+7-1ubuntu1~24.04
sudo apt install openjdk-25-jdk-headless # version 25.0.1+8-1~24.04
sudo apt install openjdk-8-jdk-headless  # version 8u472-ga-1~24.04
finn@finn-VMware-Virtual-Platform:~$ java --version
Command 'java' not found, but can be installed with:
sudo apt install openjdk-17-jre-headless # version 17.0.17+10-1~24.04, or
sudo apt install openjdk-21-jre-headless # version 21.0.9+10-1~24.04
sudo apt install default-jre             # version 2:1.17-75
sudo apt install openjdk-19-jre-headless # version 19.0.2+7-4
sudo apt install openjdk-20-jre-headless # version 20.0.2+9-1
sudo apt install openjdk-22-jre-headless # version 22~22ea-1
sudo apt install openjdk-11-jre-headless # version 11.0.29+7-1ubuntu1~24.04
sudo apt install openjdk-25-jre-headless # version 25.0.1+8-1~24.04
sudo apt install openjdk-8-jre-headless  # version 8u472-ga-1~24.04
finn@finn-VMware-Virtual-Platform:~$ gcc --version
Command 'gcc' not found, but can be installed with:
sudo apt install gcc
finn@finn-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
finn@finn-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.
finn@finn-VMware-Virtual-Platform:~$ S
```

Python en bash zijn geïnstalleerd

### Assignment 4.3: Compile

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

Fib.c

Fibonacci.java

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

Fib.c want de GCC compiler maakt er een binaire uitvoerbare code van die machine code bevat.

Which source code files are compiled to byte code?

Java. Die compileren naar .class files die bytecode bevatten en dat wordt uitgevoerd door de Java Virtual Machine

Which source code files are interpreted by an interpreter?

Python en Bash

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

C omdat die direct naar machine code compileert en direct uitgevoerd kan worden door de processor.

How do I run a Java program?

Eerst compileer je javac Filename.java

Daarna run je: java Filename.

In ons geval is dat dus: javac Fibonacci.java

Daarna run je: java Fibonacci.java

How do I run a Python program?

Python3 fib.py

How do I run a C program?

Gcc fib.c -o fib.c (outputname)

./fib.c(outputname)

How do I run a Bash script?

Chmod +x fib.sh

./fib.sh

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

C: Ja er word een executable file gemaakt met de naam die je er aan hebt gegeven.

Java: Ja er word een .class file gemaakt

Python en Bash: Nee, je source code wordt direct uitgevoerd. Er word misschien een \_\_pycache\_\_ folder gemaakt maar die voer je niet uit.

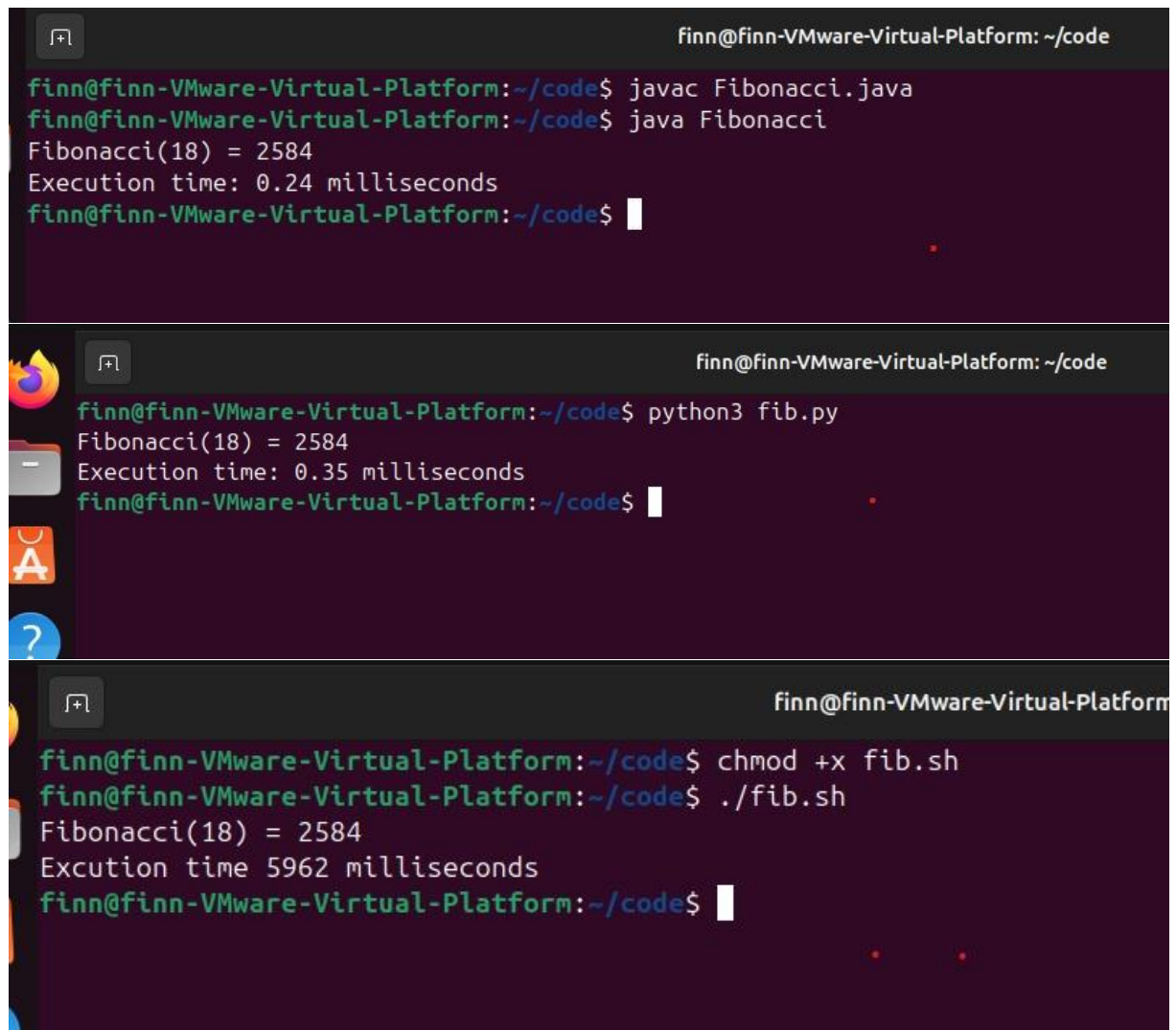
Take relevant screenshots of the following commands:

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

C is het snelste by far.

A terminal window with a dark background and light-colored text. The window title is "finn@finn-VMware-Virtual-Platform: ~/code". The terminal shows the following commands and output:

```
finn@finn-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib.c
gcc: fatal error: input file 'fib.c' is the same as output file
compilation terminated.
finn@finn-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib_c
finn@finn-VMware-Virtual-Platform:~/code$ ./fib_c
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
finn@finn-VMware-Virtual-Platform:~/code$
```



The image displays three terminal windows from a desktop environment, each showing the execution of a Fibonacci program. The first window shows the Java version, the second shows the Python version, and the third shows the Shell version. All three versions calculate the 18th Fibonacci number as 2584.

```
finn@finn-VMware-Virtual-Platform: ~/code
finn@finn-VMware-Virtual-Platform:~/code$ javac Fibonacci.java
finn@finn-VMware-Virtual-Platform:~/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.24 milliseconds
finn@finn-VMware-Virtual-Platform:~/code$
```

```
finn@finn-VMware-Virtual-Platform: ~/code
finn@finn-VMware-Virtual-Platform:~/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.35 milliseconds
finn@finn-VMware-Virtual-Platform:~/code$
```

```
finn@finn-VMware-Virtual-Platform: ~/code
finn@finn-VMware-Virtual-Platform:~/code$ chmod +x fib.sh
finn@finn-VMware-Virtual-Platform:~/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 5962 milliseconds
finn@finn-VMware-Virtual-Platform:~/code$
```

## 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.

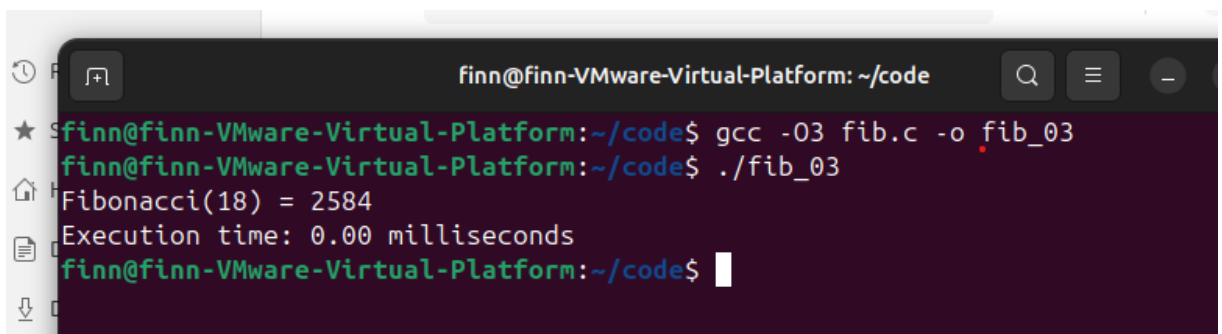
De bekende optimalisatievlaggen van `gcc` zijn:

Vlag	Betekenis
<code>-O0</code>	Geen optimalisatie (standaard)
<code>-O1</code>	Basis optimalisatie
<code>-O2</code>	Meer optimalisaties zonder extra compileertijd
<code>-O3</code>	Maximale snelheid, gebruik van meer CPU-intensieve technieken
<code>-Os</code>	Optimaliseer voor kleine bestandsgrootte
<code>-Ofast</code>	Zoals <code>-O3</code> , maar zonder naleving van bepaalde IEEE/ISO-regels (sneller, minder nauwkeurig)

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

Compile te zien in screenshot bij C

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



```
finn@finn-VMware-Virtual-Platform: ~/code
finn@finn-VMware-Virtual-Platform:~/code$ gcc -O3 fib.c -o fib_03
finn@finn-VMware-Virtual-Platform:~/code$ ./fib_03
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds
finn@finn-VMware-Virtual-Platform:~/code$
```

- 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.

```
finn@finn-VMware-Virtual-Platform: ~/code
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.38 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.55 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 9408 milliseconds

finn@finn-VMware-Virtual-Platform:~/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
```

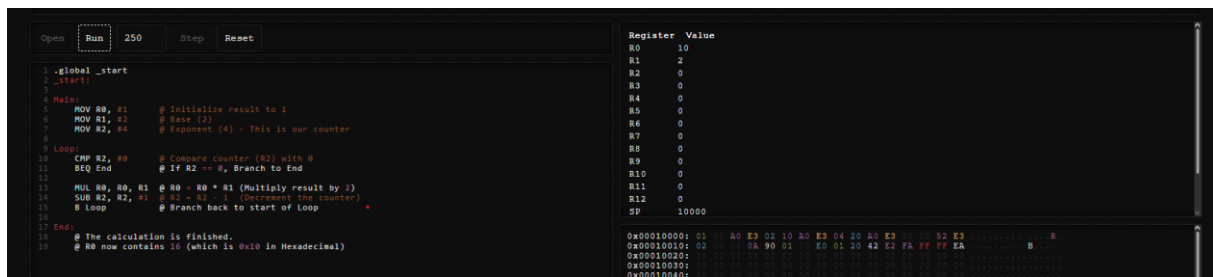
```
mov r2, #4
```

Loop:

End:

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

Screenshot of the completed code here.



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