

# Template Week 4 – Software

Student number: 592801

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the Oxd8sim debugger interface. On the left, there is a code editor window containing ARM assembly code. The code defines a function named Main that initializes registers R1 and R2, loops until R2 is zero, and calculates the factorial of R2. On the right, there is a register dump window showing the values of all registers from R0 to R10. Below the register dump is a memory dump window showing the memory starting at address 0x00010000. The memory dump shows the assembly code being executed, with the instruction pointer (IP) at address 0x00010000. The assembly code consists of several lines of ARM instructions, including mov, add, sub, and cmp operations.

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

gcc --version

python3 --version

bash --version

The screenshot shows a terminal window with a dark theme. The user is running a series of commands to check the versions of different programming languages. The commands and their outputs are as follows:

- javac --version: Output: javac 21.0.8
- java --version: Output: openjdk 21.0.8 2025-07-15  
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu124.04.1)  
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)
- gcc --version: Output: 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.
- python3 --version: Output: Python 3.12.3
- bash --version: Output: 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>>

At the bottom of the terminal window, there is a message: "This is free software; you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law."

### **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

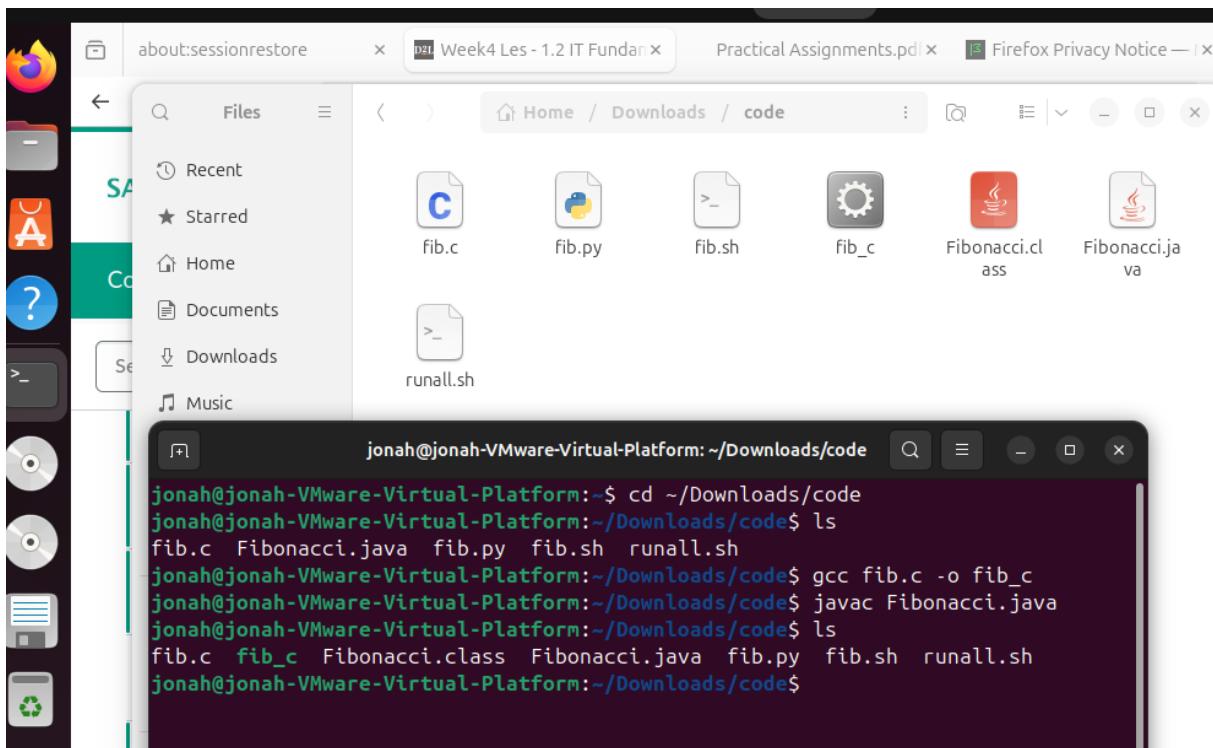
If I compile the above source code, will a new file be created? If so, which file?

Fib.c wordt fib\_c en Fibonacci.java word 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?

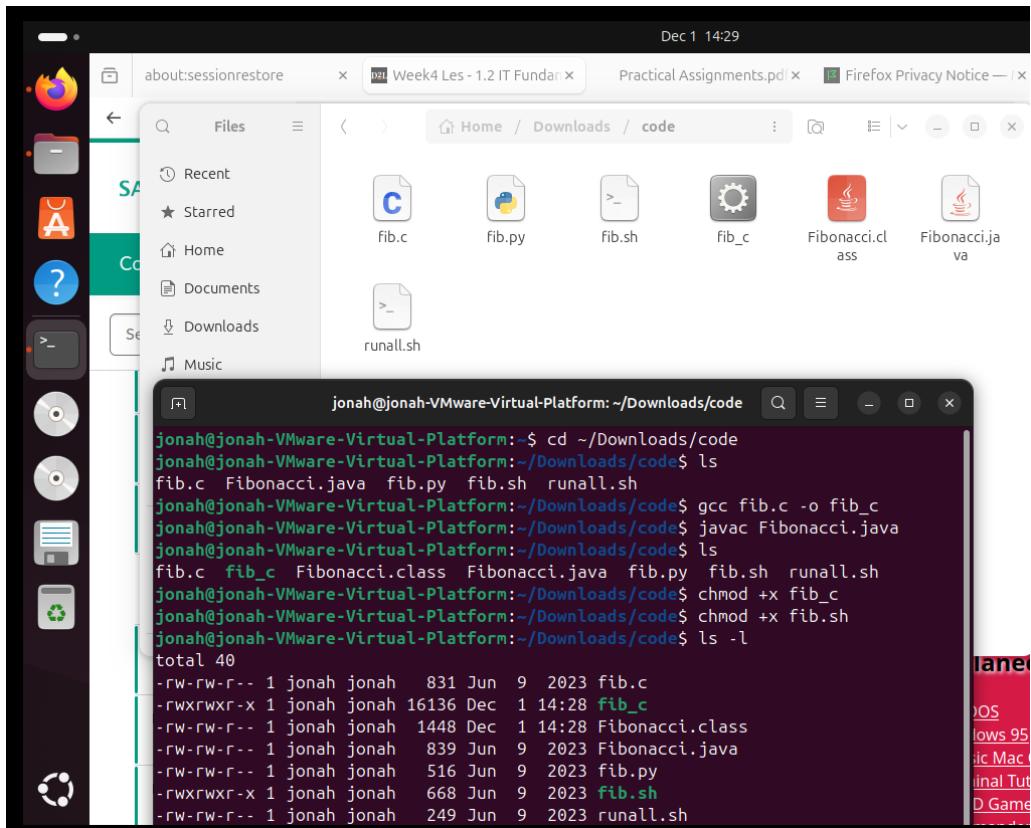
## Compiling the files



The screenshot shows a Linux desktop environment with a file manager window open in the foreground. The file manager displays a folder structure under 'Downloads/code' containing several files: fib.c, fib.py, fib.sh, fib\_c, Fibonacci.class, Fibonacci.java, and runall.sh. In the background, a terminal window is running on a virtual machine named 'VMware-Virtual-Platform'. The terminal session shows the user navigating to the directory, listing the files, and then compiling them using gcc and javac. The terminal output is as follows:

```
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -o fib_c
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c fib_c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$
```

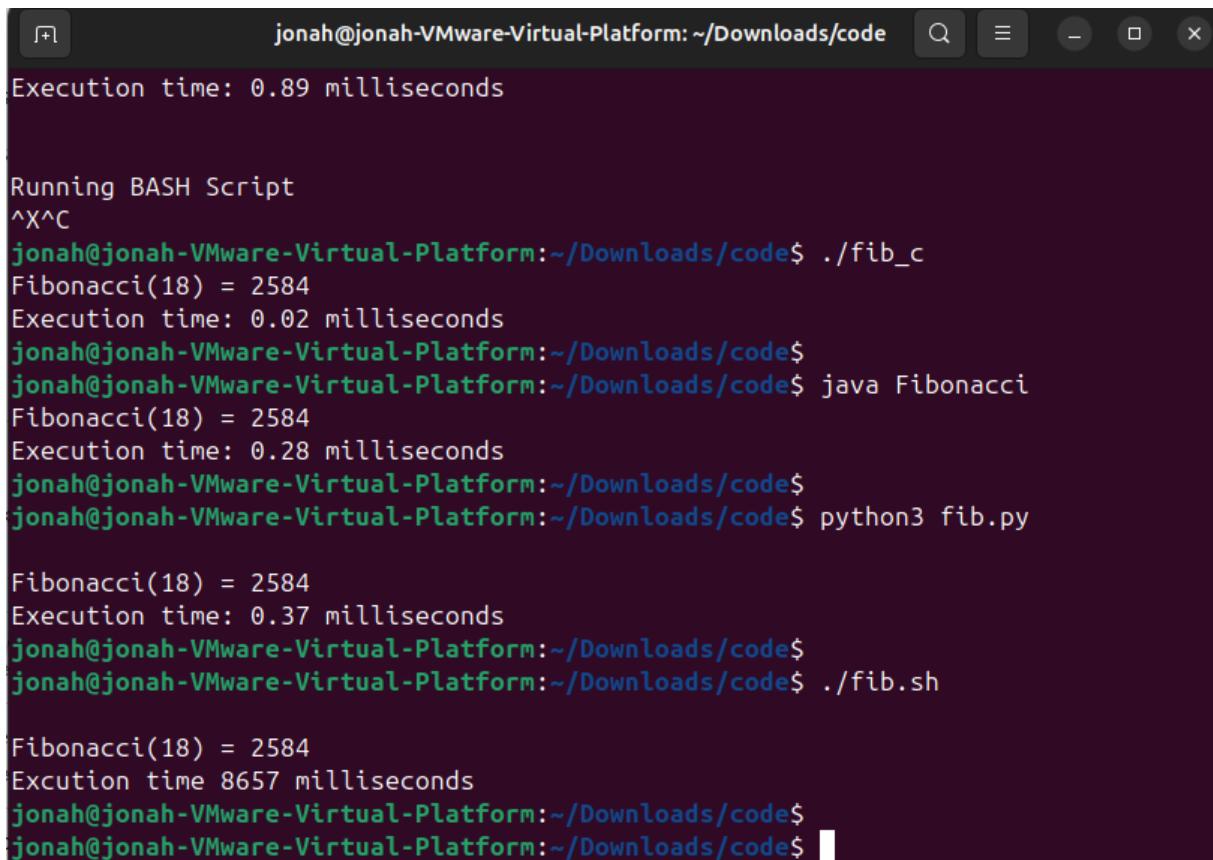
## Executable maken



The screenshot shows a Linux desktop environment with a file manager window open in the foreground. The file manager displays a folder structure under 'Downloads/code' containing files: fib.c, fib.py, fib.sh, fib\_c, Fibonacci.class, Fibonacci.java, and runall.sh. In the background, a terminal window is running on a virtual machine named 'VMware-Virtual-Platform'. The terminal session shows the user navigating to the directory, listing the files, and then compiling them using gcc and javac. After compilation, the user runs chmod +x on the executables fib\_c and fib.sh. Finally, they use ls -l to list all files again, showing the executable permissions. The terminal output is as follows:

```
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -o fib_c
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c fib_c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ chmod +x fib_c
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ chmod +x fib.sh
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ ls -l
total 40
-rw-rw-r-- 1 jonah jonah 831 Jun  9 2023 fib.c
-rwxrwxr-x 1 jonah jonah 16136 Dec  1 14:28 fib_c
-rw-rw-r-- 1 jonah jonah 1448 Dec  1 14:28 Fibonacci.class
-rw-rw-r-- 1 jonah jonah 839 Jun  9 2023 Fibonacci.java
-rw-rw-r-- 1 jonah jonah 516 Jun  9 2023 fib.py
-rwxrwxr-x 1 jonah jonah 668 Jun  9 2023 fib.sh
-rw-rw-r-- 1 jonah jonah 249 Jun  9 2023 runall.sh
```

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



The screenshot shows a terminal window with the following session:

```
jonah@jonah-Virtual-Platform:~/Downloads/code$ ./fib_c
Execution time: 0.89 milliseconds

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

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

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

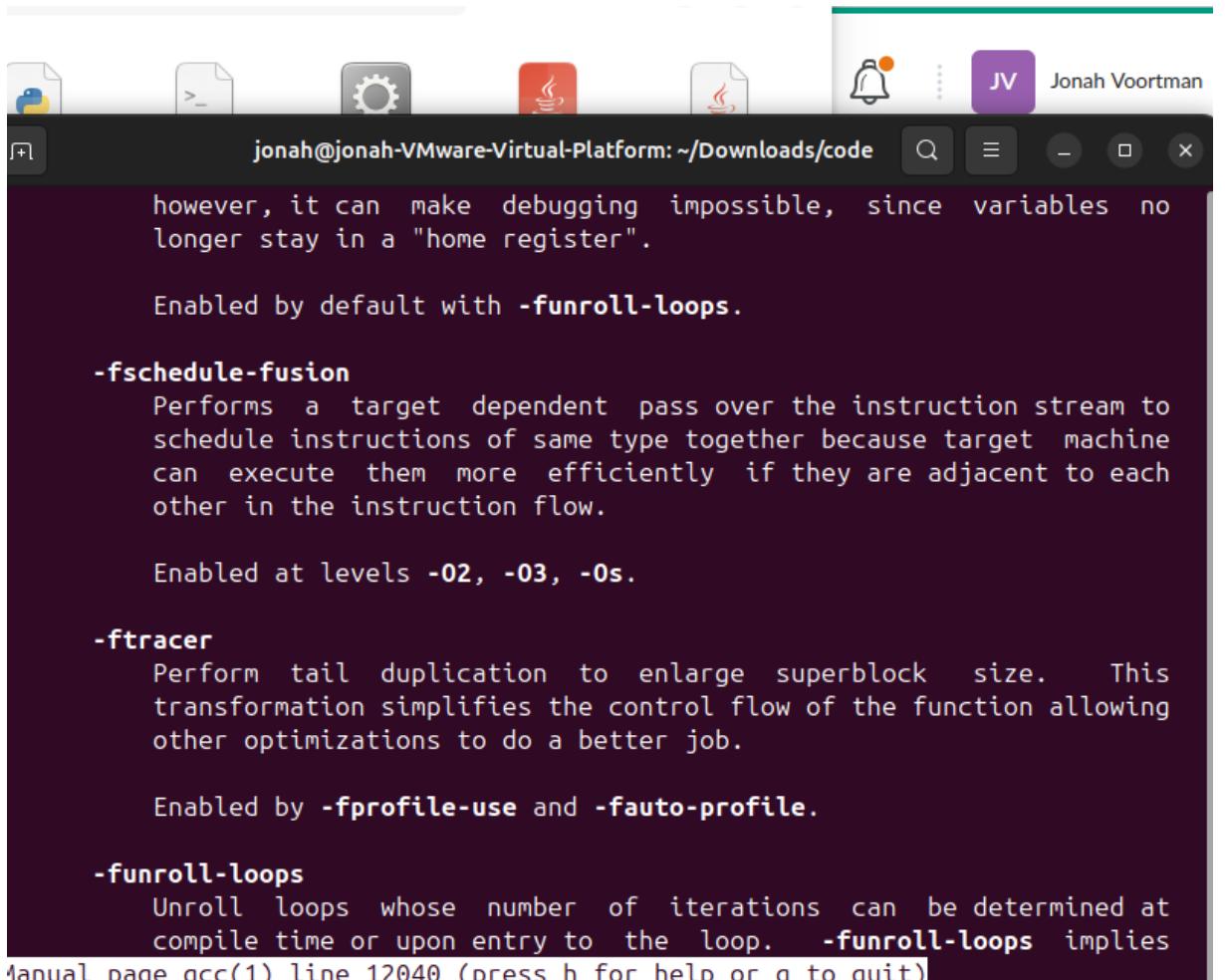
#### Assignment 4.4: Optimize

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:

- 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 with a dark theme. At the top, there are several icons: a Python logo, a file with a plus sign, a file with a minus sign, a gear, a coffee cup, another file, a bell, and a purple square with 'JV'. To the right of these is the name 'Jonah Voortman'. Below the icons, the terminal title bar reads 'jonah@jonah-Virtual-Platform: ~/Downloads/code'. The main area of the terminal contains text from the GCC documentation:

```
however, it can make debugging impossible, since variables no longer stay in a "home register".  
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

A screenshot of a Linux desktop environment. In the foreground, a terminal window is open with the command line: `jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$`. The terminal displays the following output:

```
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ man gcc
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ man gcc
[1]+ Stopped man gcc
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ man gcc
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ gcc -O2 fib.c -o fib_c_opt
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds

real 0m0.002s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real 0m0.001s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c_opt
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds

real 0m0.002s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real 0m0.001s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$
```

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

A screenshot of a Linux desktop environment, identical to the one above. In the foreground, a terminal window is open with the command line: `jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$`. The terminal displays the following output:

```
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ man gcc
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ man gcc
[1]+ Stopped man gcc
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ man gcc
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ gcc -O2 fib.c -o fib_c_opt
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds

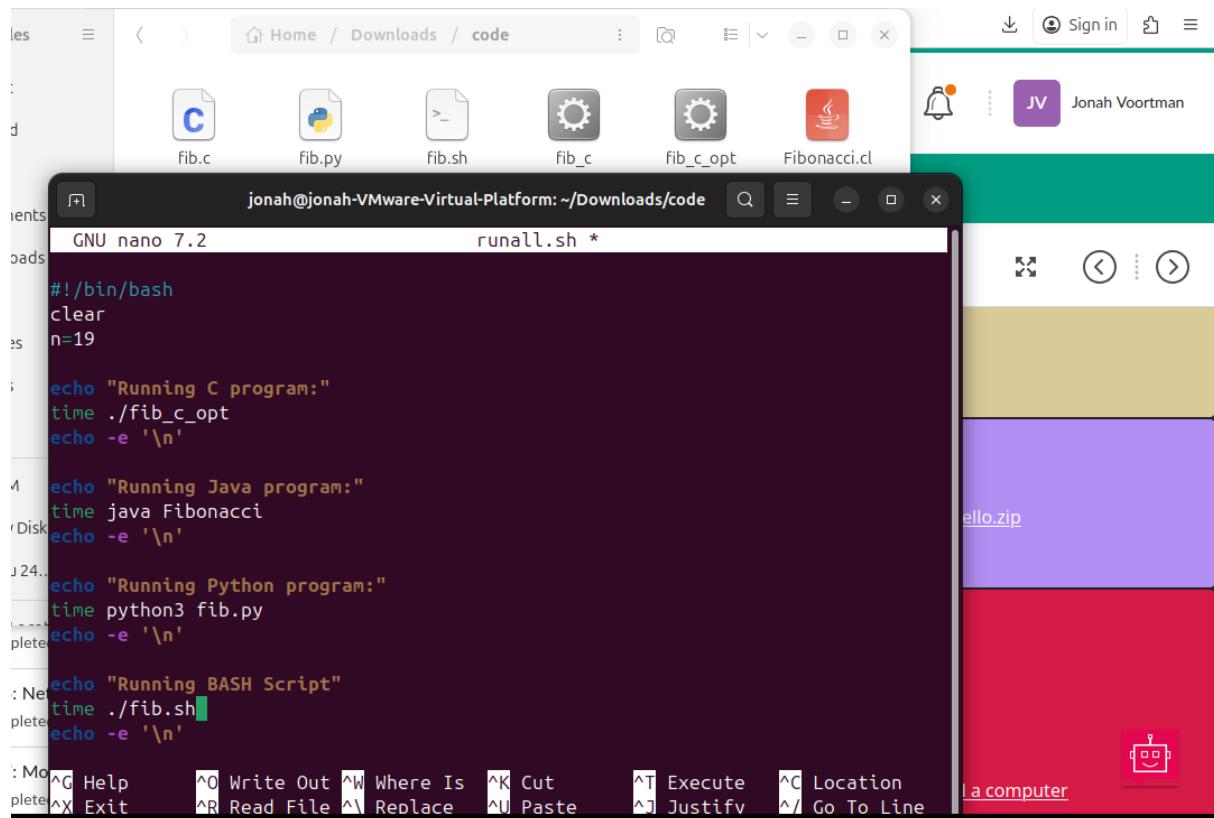
real 0m0.002s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real 0m0.001s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c_opt
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds

real 0m0.002s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/code$ time ./fib_c_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real 0m0.001s
user 0m0.000s
sys 0m0.002s
jonah@jonah-VMware-Virtual-Platform:~/Downloads/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.



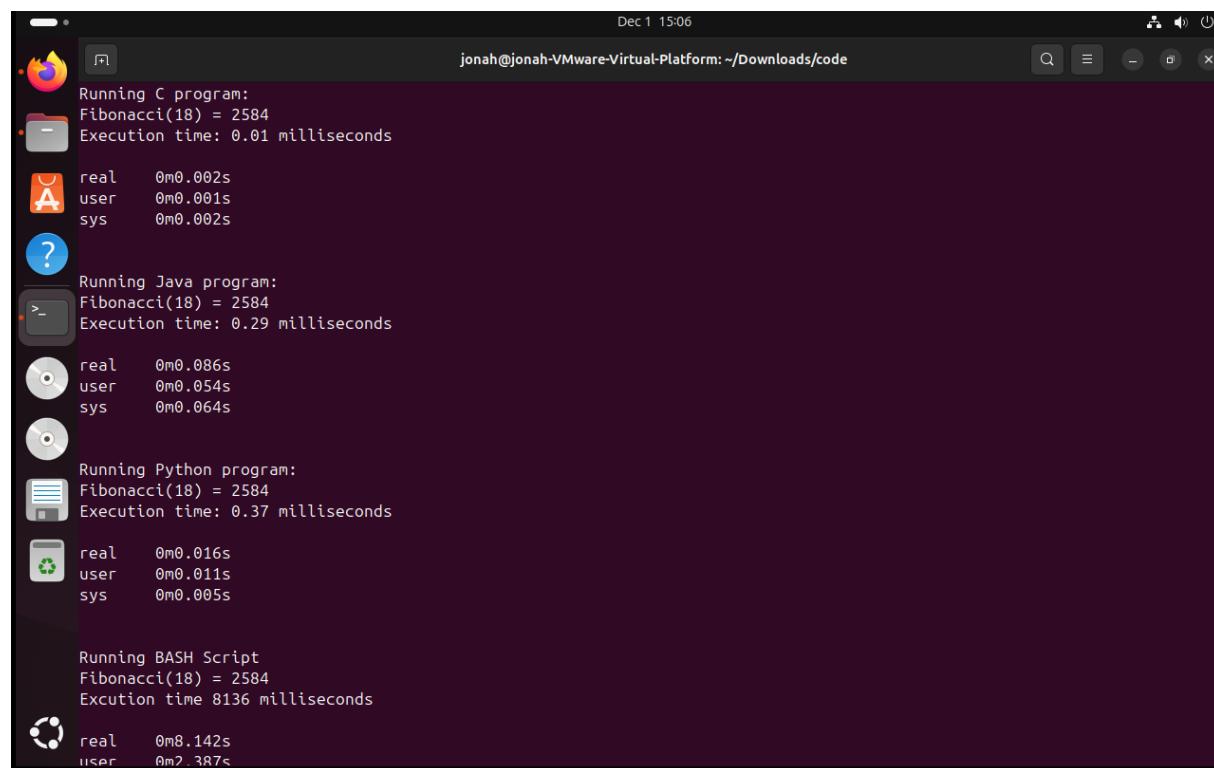
```
GNU nano 7.2 runall.sh *
#!/bin/bash
clear
n=19

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'
```



```
jonah@jonah-VMware-Virtual-Platform: ~/Downloads/code
Dec 1 15:06

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.

The screenshot shows the OakSim debugger interface. The assembly code window contains the following code:

```
1 Main:  
2 mov r0, #1  
3 mov r1, #2  
4 mov r2, #4  
5  
6 Loop:  
7 mul r0, r0, r1  
8 sub r2, r2, #1  
9 cmp r2, #0  
10 beq End  
11 b Loop  
12  
13 End:  
14
```

The register window shows the following values:

Register	Value
R0	10
R1	2
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0

The memory dump window shows the memory starting at address 0x00010000. The first few bytes are 01 20 42 E3 02 10 A0 E3 04 20 A0 E3 90 01 00 E0, followed by a series of zeros.

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