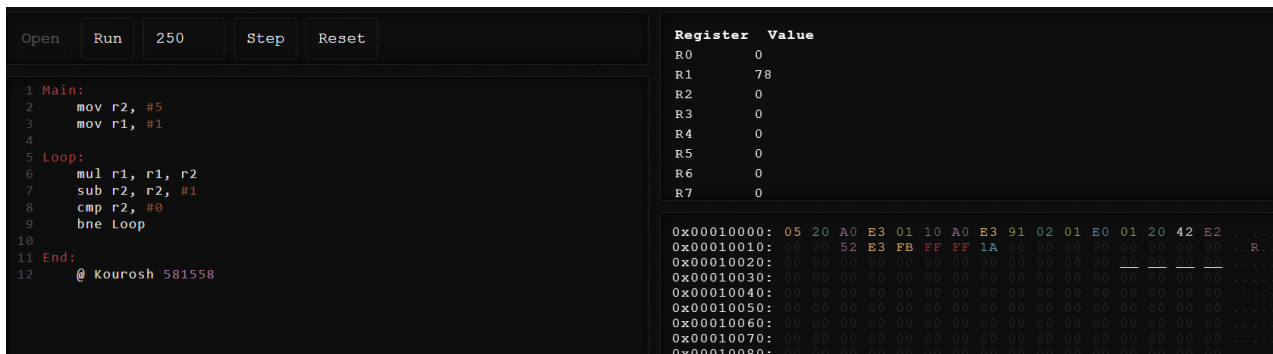


Template Week 4 – Software

Student number: 581558

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



The screenshot shows an ARM assembly editor with a code window on the left and a register window on the right. The code window contains the following assembly code:

```
1 Main:
2     mov r2, #5
3     mov r1, #1
4
5 Loop:
6     mul r1, r1, r2
7     sub r2, r2, #1
8     cmp r2, #0
9     bne Loop
10
11 End:
12     @ Kourosh 581558
```

The register window on the right shows the following values:

Register	Value
R0	0
R1	78
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

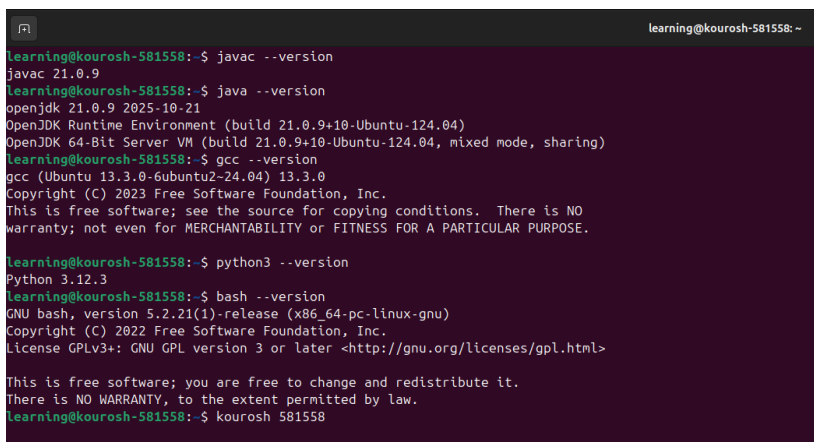
javac --version

java --version

gcc --version

python3 --version

bash --version



The screenshot shows a terminal window with the following output:

```
learning@kourosh-581558:~$ javac --version
javac 21.0.9
learning@kourosh-581558:~$ 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)
learning@kourosh-581558:~$ 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.
learning@kourosh-581558:~$ python3 --version
Python 3.12.3
learning@kourosh-581558:~$ 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.
learning@kourosh-581558:~$ kourosh 581558
```

Assignment 4.3: Compile

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

fib.c and Fibonacci.java

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

fib.c

Which source code files are compiled to byte code?

Fibonacci.java

Which source code files are interpreted by an interpreter?

fib.py

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?

- javac Fibonacci.java

- java Fibonacci

How do I run a Python program?

- python3 fib.py

How do I run a C program?

- gcc fib.c -o fibProgram

- fibProgram.exe (in windows) or ./fibProgram (in linux)

How do I run a Bash script?

- chmod a+x fib.sh

- bash fib.sh

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

Gcc → Yes, it creates an executable file

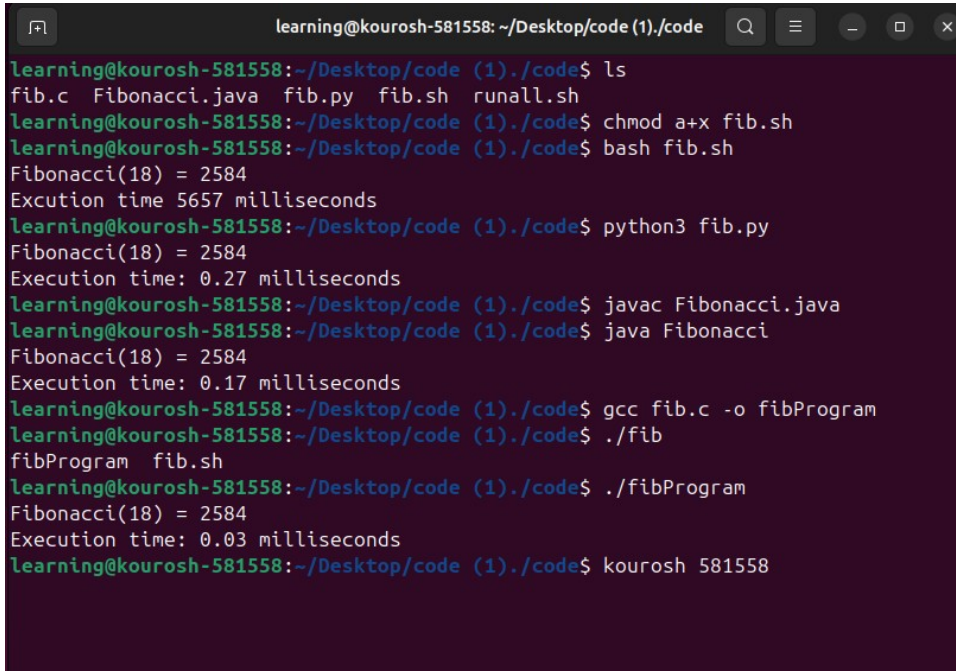
Javac → Yes, it creates a bytecode file

Python3 → No

Bash → No

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?



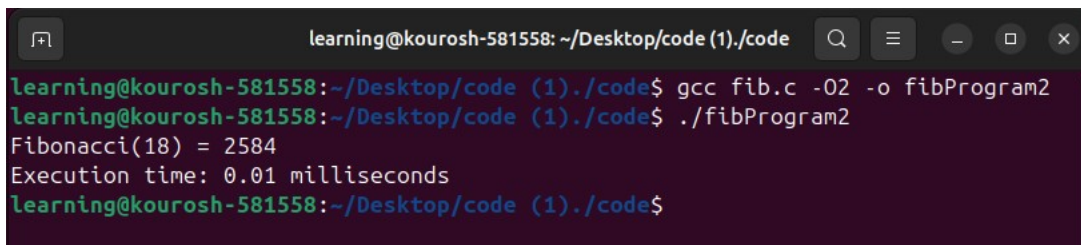
```
learning@kourosch-581558: ~/Desktop/code (1)/code
learning@kourosch-581558:~/Desktop/code (1)/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
learning@kourosch-581558:~/Desktop/code (1)/code$ chmod a+x fib.sh
learning@kourosch-581558:~/Desktop/code (1)/code$ bash fib.sh
Fibonacci(18) = 2584
Execution time 5657 milliseconds
learning@kourosch-581558:~/Desktop/code (1)/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
learning@kourosch-581558:~/Desktop/code (1)/code$ javac Fibonacci.java
learning@kourosch-581558:~/Desktop/code (1)/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.17 milliseconds
learning@kourosch-581558:~/Desktop/code (1)/code$ gcc fib.c -o fibProgram
learning@kourosch-581558:~/Desktop/code (1)/code$ ./fib
fibProgram  fib.sh
learning@kourosch-581558:~/Desktop/code (1)/code$ ./fibProgram
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
learning@kourosch-581558:~/Desktop/code (1)/code$ kourosch 581558
```

GCC compiled it the fastest with only 0.03 milliseconds.

Assignment 4.4: Optimize

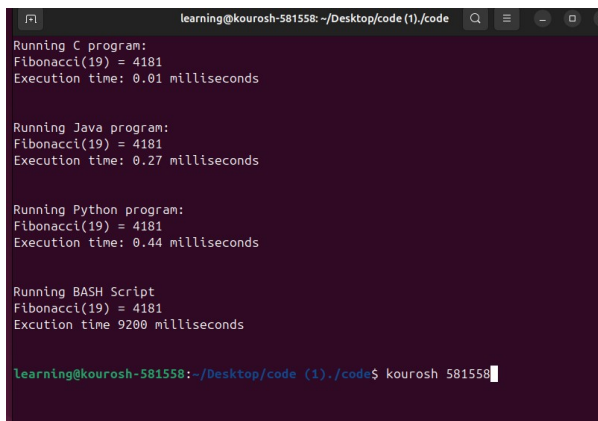
Take relevant screenshots of the following commands:

- 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.
- Compile **fib.c** again with the optimization parameters
- Run the newly compiled program. Is it true that it now performs the calculation faster?
Yes, O2 optimization reduced its execution time by 0.02 milliseconds from 0.03 ms to 0.01 ms



```
learning@kourosch-581558: ~/Desktop/code (1)/code
learning@kourosch-581558:~/Desktop/code (1)/code$ gcc fib.c -O2 -o fibProgram2
learning@kourosch-581558:~/Desktop/code (1)/code$ ./fibProgram2
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
learning@kourosch-581558:~/Desktop/code (1)/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.



```
learning@kourosch-581558: ~/Desktop/code (1)/code
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.27 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.44 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 9200 milliseconds

learning@kourosch-581558:~/Desktop/code (1)/code$ kourosh 581558
```

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 an ARM assembly simulator interface. On the left, there are buttons for 'Open', 'Run', '250', 'Step', and 'Reset'. Below these is a list of assembly instructions with line numbers:

```
1 Main:
2   mov r1, #2
3   mov r2, #4
4   mov r0, #1
5
6 Loop:
7   mul r0, r0, r1
8   subs r2, r2, #1
9   bne Loop
10
11 End:
12   @ Kourosh 581558
13
```

On the right, there is a 'Register Value' table showing the current state of the registers:

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

Below the register table, there is a memory dump showing hexadecimal values for addresses from 0x00010000 to 0x00010080.

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