Week 4 – Software

Student number: 573534

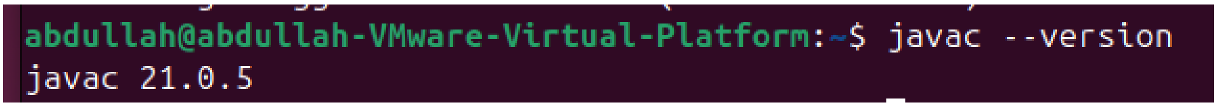
Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

Assignment 4.2: Programming languages

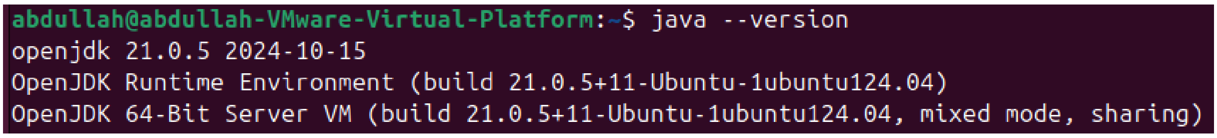
Take screenshots that the following commands work:

javac –version



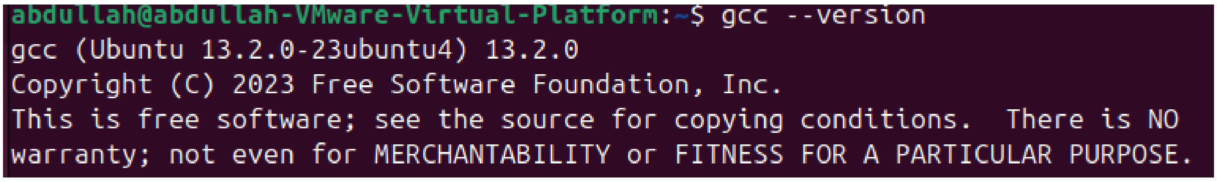
```
abdullah@abdullah-VMware-Virtual-Platform:~$ javac --version
javac 21.0.5
```

java –version



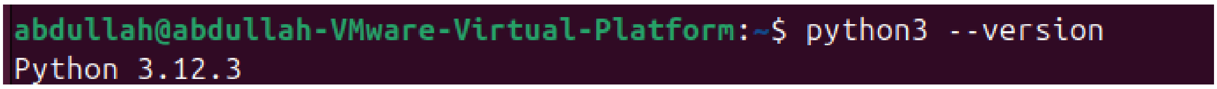
```
abdullah@abdullah-VMware-Virtual-Platform:~$ java --version
openjdk 21.0.5 2024-10-15
OpenJDK Runtime Environment (build 21.0.5+11-Ubuntu-1ubuntu124.04)
OpenJDK 64-Bit Server VM (build 21.0.5+11-Ubuntu-1ubuntu124.04, mixed mode, sharing)
```

gcc –version



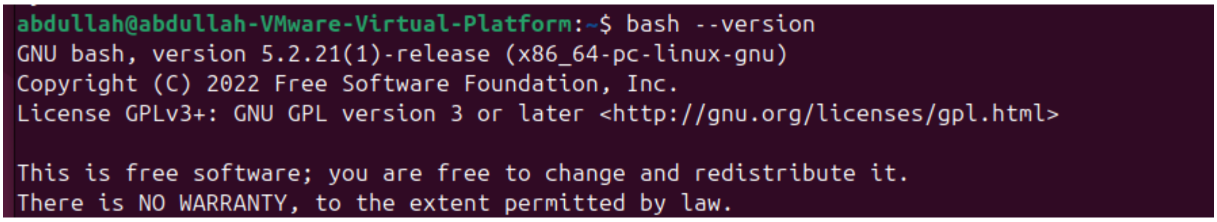
```
abdullah@abdullah-VMware-Virtual-Platform:~$ gcc --version
gcc (Ubuntu 13.2.0-23ubuntu4) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

python3 –version



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

bash –version



```
abdullah@abdullah-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.
```

Assignment 4.3: Compile

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

- Fibonacci.java
- fib.c

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
- 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 will perform the calculation the fastest. This is because the code is compiled directly into machine code, which is executed natively by the processor.

How do I run a Java program?

- First you need to go to the directory of the java file.
- Then execute “javac java_file.java” in a command prompt.
- Then execute “java java_file” in a command prompt to run the program.

How do I run a Python program?

- First you need to go to the directory of the python file.
- Then execute “python3 python_file.py” in a command prompt.

How do I run a C program?

- First you need to go to the directory of the C file.
- Then execute “gcc -o c_program c_program.c” in a command prompt.
- Then execute “./c_program”

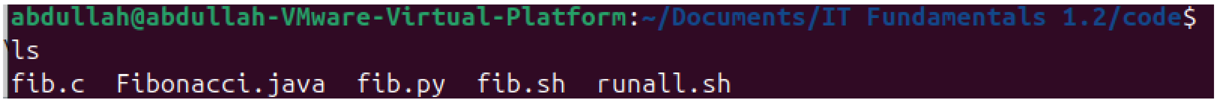
How do I run a Bash script?

- chmod +x bash_script.sh
- ./bash_script.sh

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

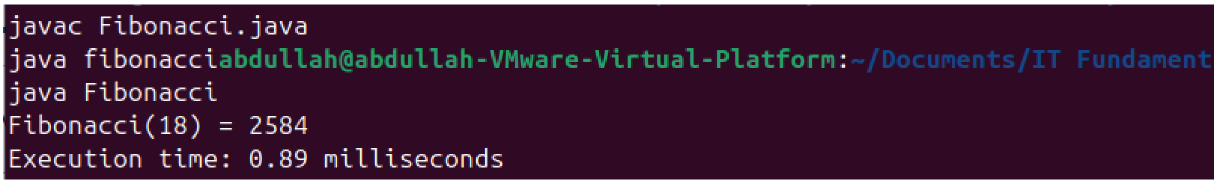
- Yes, for the C code a new file will be created called “c_program.exe”
- Yes, for the Java code a new file will be created called “java_file.class”

Take relevant screenshots of the following commands:

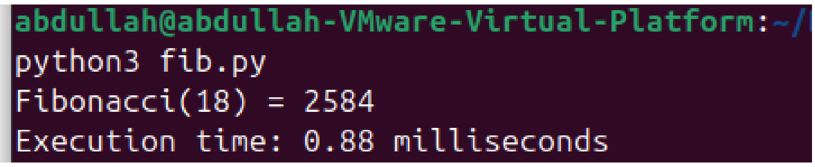


```
abdullah@abdullah-VMware-Virtual-Platform:~/Documents/IT Fundamentals 1.2/code$
ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
```

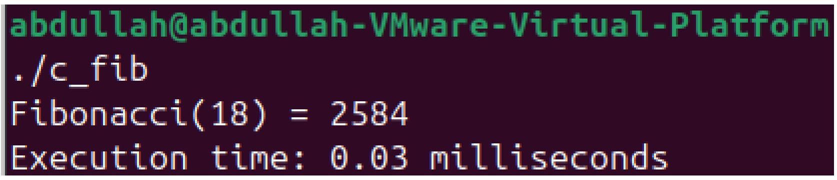
- Compile the source files where necessary
  - Java
  - C
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?



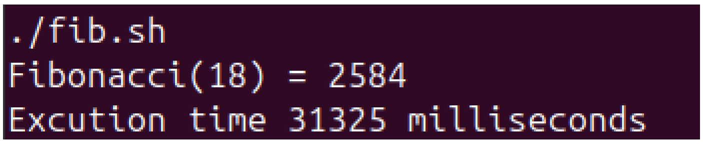
```
javac Fibonacci.java
java fibonacciabdullah@abdullah-VMware-Virtual-Platform:~/Documents/IT Fundament
java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.89 milliseconds
```



```
abdullah@abdullah-VMware-Virtual-Platform:~/
python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.88 milliseconds
```



```
abdullah@abdullah-VMware-Virtual-Platform
./c_fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
```



```
./fib.sh
Fibonacci(18) = 2584
Excution time 31325 milliseconds
```

The C file is the fastest and the bash script is the slowest.

Assignment 4.4: Optimize

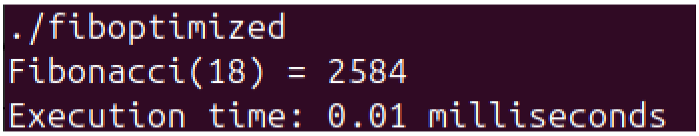
Take relevant screenshots of the following commands:

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.

   a. The optimizations I have added to my code to make the execution time faster is -O3 and -march=native.
   -O3: Maximizes optimization, including loop unrolling, function inlining, and vectorization.
   -march=native: Leverages all the capabilities of the host CPU, ensuring the compiled code takes advantage of features like SIMD or AVX if available.

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

   a.



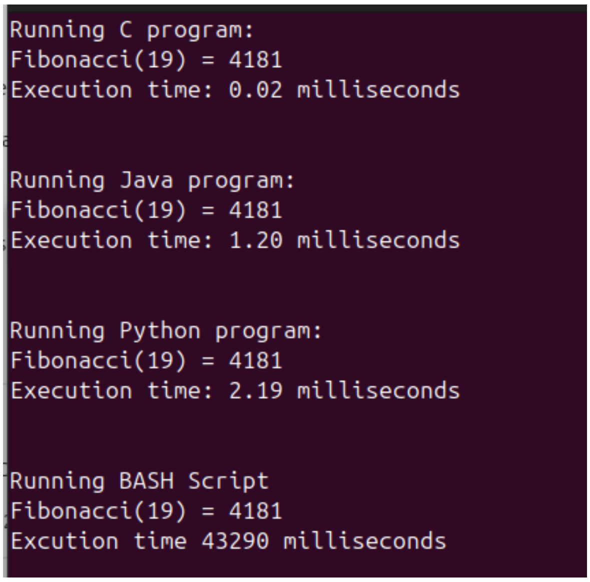
```
./fiboptimized
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
```

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

   a. Yes it performs the calculation 3x faster, now the execution time is 0.01 instead of 0.03ms.

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.

   a.

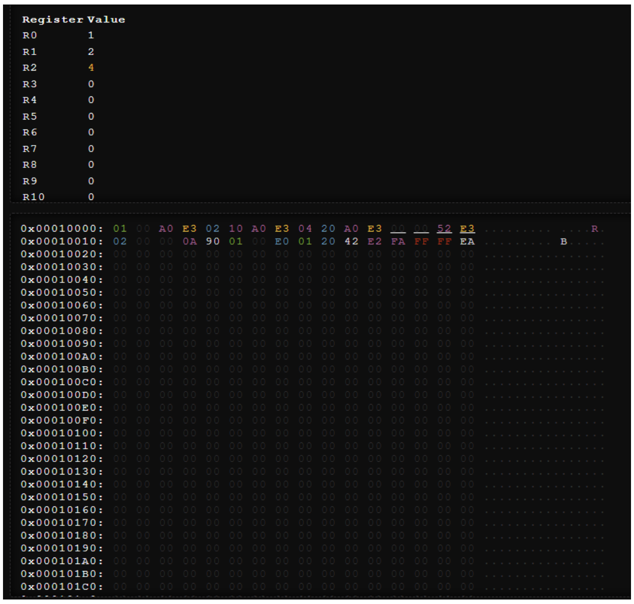


```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.02 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 1.20 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 2.19 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 43290 milliseconds
```

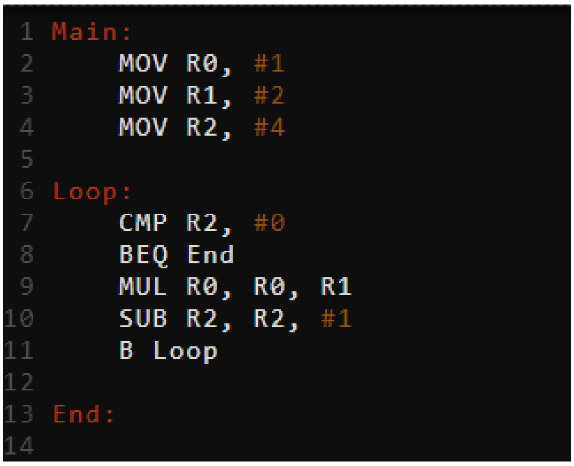
Bonus point assignment – week 4

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 R0, #1 // Zet nummer 1 op R0

    MOV R1, #2 // Zet nummer 2 op R1 (dit is de base)

    MOV R2, #4 // Zet nummer 4 op R2 (dit is het aantal macht)


Loop:

    CMP R2, #0    // Check of R2 0 is

    BEQ End      // Als dat zo is stop je de LOOP!

    MUL R0, R0, R1          // R0 word R0 * R1

    SUB R2, R2, #1          // R2 word R2 - 1

    B Loop             // LOOP!


End:
```



```
Register Value
R0       1
R1       2
R2       4
R3       0
R4       0
R5       0
R6       0
R7       0
R8       0
R9       0
R10      0

0x00010000: 01 00 A0 E3 02 10 A0 E3 04 20 A0 E3 00 00 52 E3 ......... ....R.
0x00010010: 02 00 00 0A 90 01 00 E0 01 20 42 E2 FA FF FF EA ......... B.....
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 ................
0x000100B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000101A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000101B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000101C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```



```
Main:
    MOV R0, #1
    MOV R1, #2
    MOV R2, #4

Loop:
    CMP R2, #0
    BEQ End
    MUL R0, R0, R1
    SUB R2, R2, #1
    B Loop

End:
```