

Template Week 4 – Software

Student number: 578688

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim software interface. On the left, the assembly code for calculating a factorial is displayed:

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

On the right, the register values and memory dump are shown. The registers are:

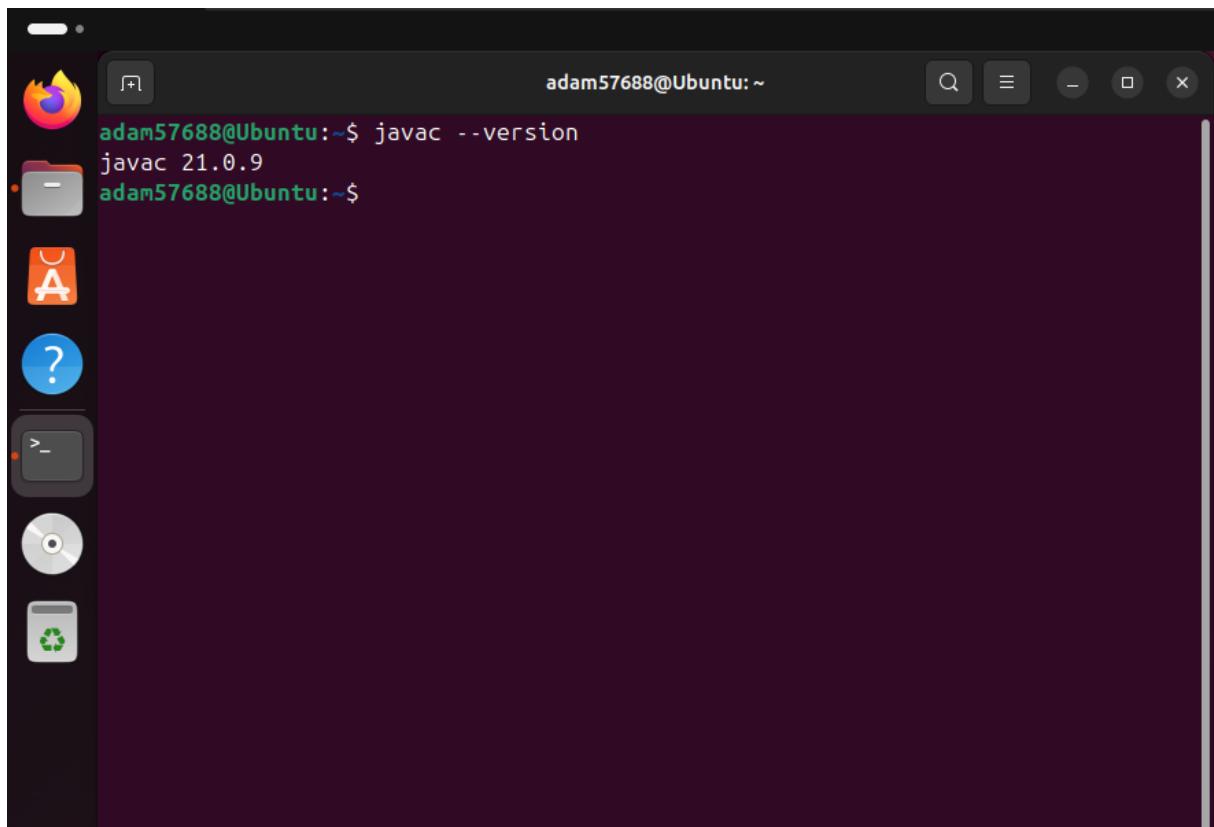
Register	Value
R0	1
R1	78
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0

The memory dump shows the stack starting at address 0x000010000, containing the assembly code and some initial values. The stack grows downwards.

Assignment 4.2: Programming languages

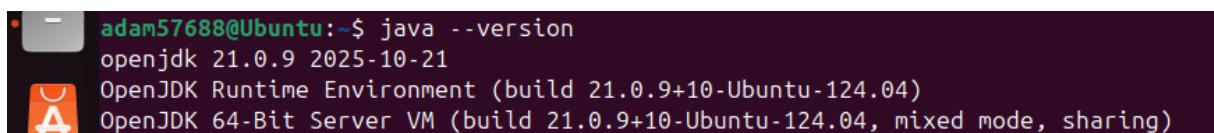
Take screenshots that the following commands work:

javac --version



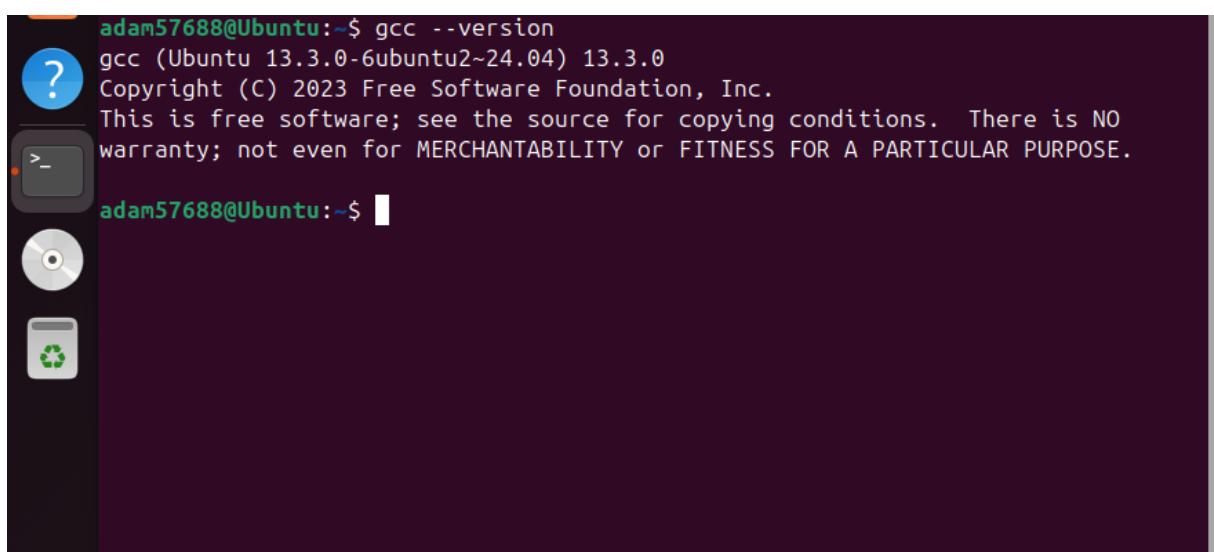
```
adam57688@Ubuntu:~$ javac --version
javac 21.0.9
adam57688@Ubuntu:~$
```

java --version



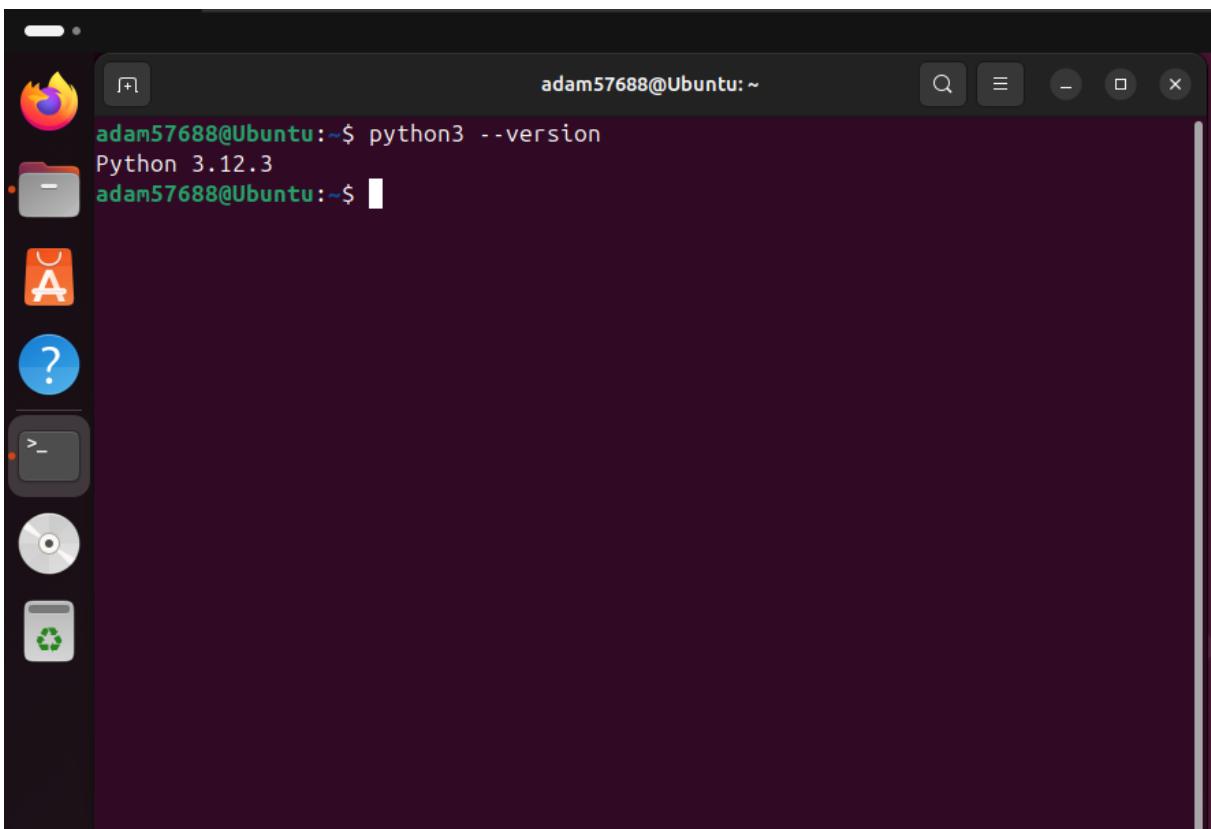
```
adam57688@Ubuntu:~$ java --version
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
```

gcc --version



```
adam57688@Ubuntu:~$ 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.
```

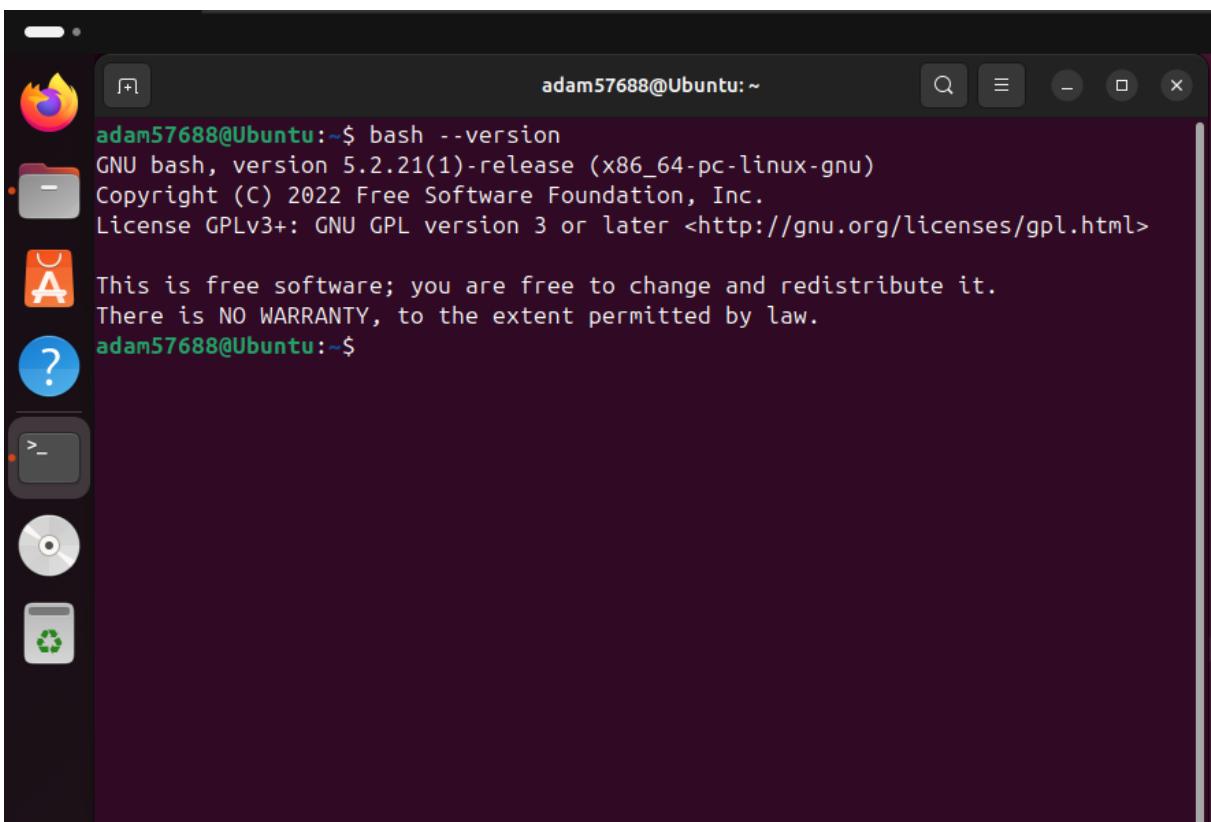
```
python3 --version
```



A screenshot of a terminal window on an Ubuntu desktop environment. The window title bar says "adam57688@Ubuntu: ~". The terminal itself shows the command "python3 --version" followed by the output "Python 3.12.3". The window has a dark purple background and a vertical scroll bar on the right. A dock on the left contains icons for various applications like a browser, file manager, terminal, and system settings.

```
adam57688@Ubuntu:~$ python3 --version
Python 3.12.3
adam57688@Ubuntu:~$
```

```
bash --version
```



A screenshot of a terminal window on an Ubuntu desktop environment. The window title bar says "adam57688@Ubuntu: ~". The terminal shows the command "bash --version" followed by the output of the GNU bash version information, including the release date (May 2022), copyright (Free Software Foundation, Inc.), and license (GPLv3+). It also includes the standard GNU free software notice about redistribution and warranty. The window has a dark purple background and a vertical scroll bar on the right. A dock on the left contains icons for various applications like a browser, file manager, terminal, and system settings.

```
adam57688@Ubuntu:~$ 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.
adam57688@Ubuntu:~$
```

Assignment 4.3: Compile

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

Java, C and python

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

C

Which source code files are compiled to byte code?

Java

Which source code files are interpreted by an interpreter?

Python

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

C

How do I run a Java program?

java filename

How do I run a Python program?

python3 file name

How do I run a C program?

./filename

How do I run a Bash script?

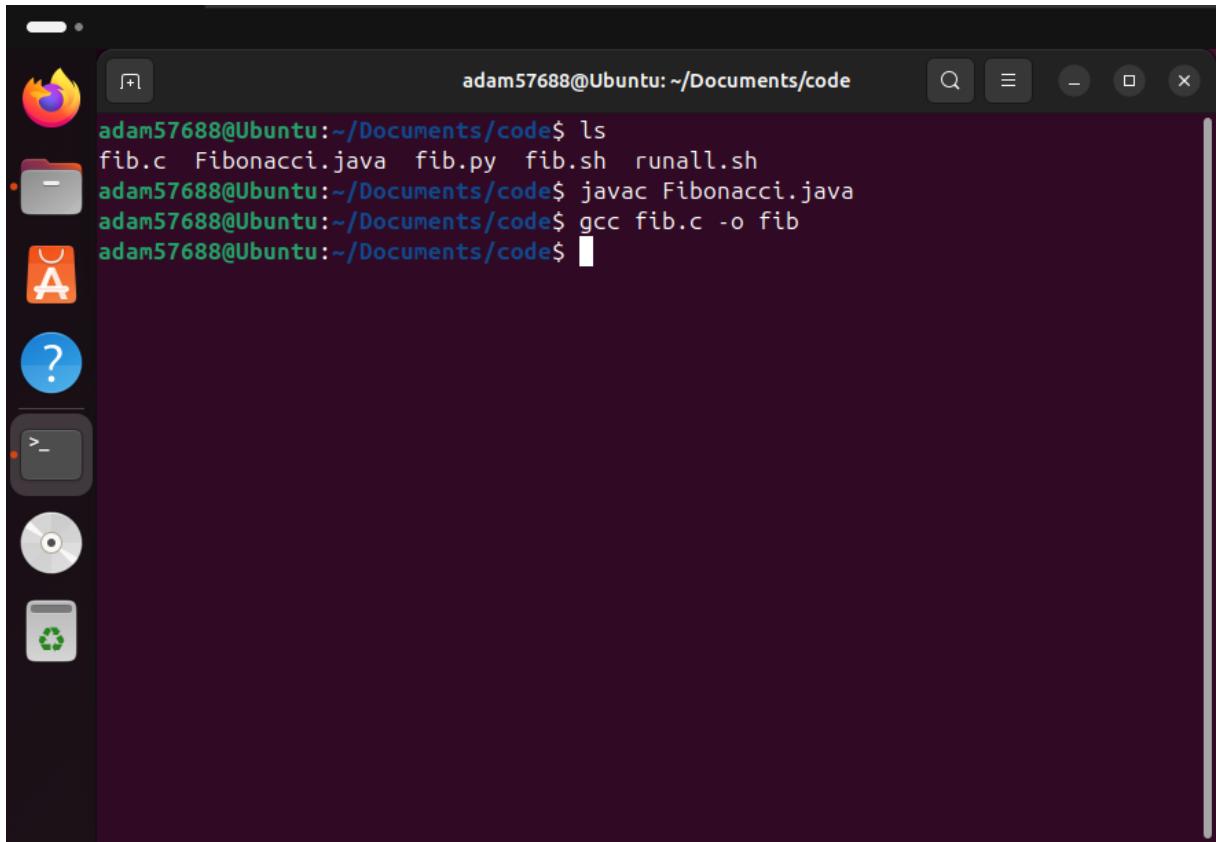
./filename

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

Yes, fib.c, Fibonacci.class, fib.py

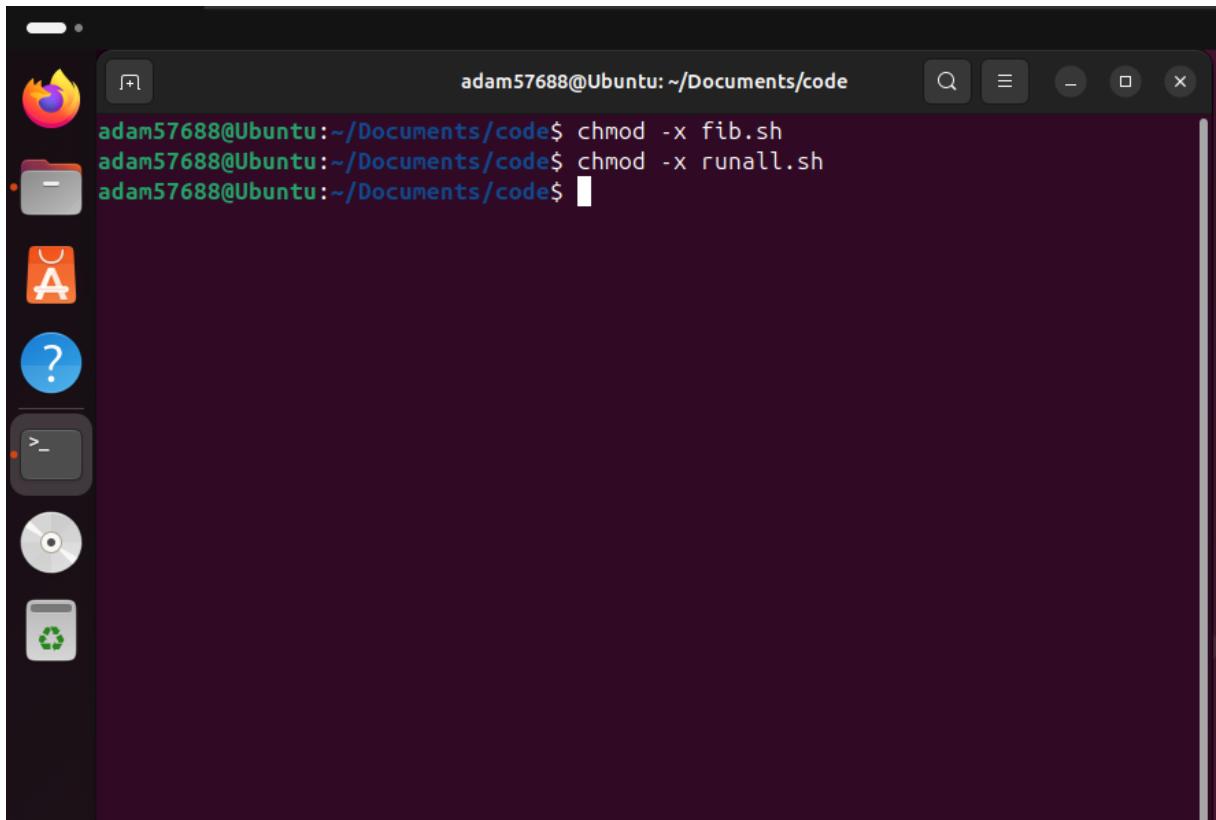
Take relevant screenshots of the following commands:

- Compile the source files where necessary



```
adam57688@Ubuntu:~/Documents/code$ ls  
fib.c Fibonacci.java fib.py fib.sh runall.sh  
adam57688@Ubuntu:~/Documents/code$ javac Fibonacci.java  
adam57688@Ubuntu:~/Documents/code$ gcc fib.c -o fib  
adam57688@Ubuntu:~/Documents/code$
```

- Make them executable



```
adam57688@Ubuntu:~/Documents/code$ chmod -x fib.sh  
adam57688@Ubuntu:~/Documents/code$ chmod -x runall.sh  
adam57688@Ubuntu:~/Documents/code$
```

- Run them

The screenshot shows a terminal window on a Linux system (Ubuntu) with a dark theme. The terminal title bar reads "adam57688@Ubuntu: ~/Documents/code". The terminal window contains the following text:

```
adam57688@Ubuntu:~/Documents/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
adam57688@Ubuntu:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
adam57688@Ubuntu:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.16 milliseconds
adam57688@Ubuntu:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.18 milliseconds
adam57688@Ubuntu:~/Documents/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 5331 milliseconds
adam57688@Ubuntu:~/Documents/code$
```

- Which (compiled) source code file performs the calculation the fastest?
- Fib.c

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.

```
gcc -O -o
```

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

```
adam57688@Ubuntu:~/Documents/code$ gcc fib.c -O3 -o fib
adam57688@Ubuntu:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
adam57688@Ubuntu:~/Documents/code$
```

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

Yes.

- 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.

```
adam57688@Ubuntu:~/Documents/code$ ./runall.sh
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.31 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.31 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Excution time 10592 milliseconds

adam57688@Ubuntu:~/Documents/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.

The screenshot shows the OakSim interface. On the left is the assembly code editor with the following content:

```
1    mov r0, #2  
2    mov r1, #4  
3  
4  
5  
6 Loop:  
7    mov r2, #2  
8    sub r1, r1, #1  
9    mul r0, r0, r2  
10   --  
11   cmp r1, #1  
12   beq Exit  
13   b Loop  
14  
15  
16  
17 Exit:
```

On the right is the register dump window titled "Registers". It lists the following registers and their values:

Register	Value
R0	10
R1	1
R2	2
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0

Below the register dump is a memory dump window showing memory starting at address 0x000010000. The memory contains the following data:

Address	Value
0x000010000	02 00 A0 E3 04 10 A0 E3 02 20 A0 E3 01 10 41 E2
0x000010010	90 02 00 E0 01 00 51 E3 00 00 00 0A F9 FF FF EA
0x000010020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010170	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000101A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000101B0	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**