

# Template Week 4 - Software

Student number: 588734

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a debugger interface with the following details:

- Assembly Code:**

```
1 Loop:  
2     add r0, r0, #1  
3     mul r1, r0, r0  
4     sub r2, r1, r0  
5     mov r3, #15
```
- Registers:**

Register	Value
R0	b
R1	79
R2	6e
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
- Memory Dump:** A hex dump of memory starting at address 0x0000100000, showing the value 01 00 80 E2 90 00 01 E0 00 20 41 E0 FB FF FF EA.

What is the value of:

- r0: b
- r1: 79
- r2: 6e
- r3: 0

The screenshot shows a debugger interface with the following details:

- Assembly Code:**

```
1 Loop:  
2     add r0, r0, #1  
3     mul r1, r0, r0  
4     cmp r1, #144  
5     beq exit  
6     b loop  
7 Exit:[
```
- Registers:**

Register	Value
R0	0
R1	0
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
- Memory Dump:** A hex dump of memory starting at address 0x0000100000, showing all zeros.

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

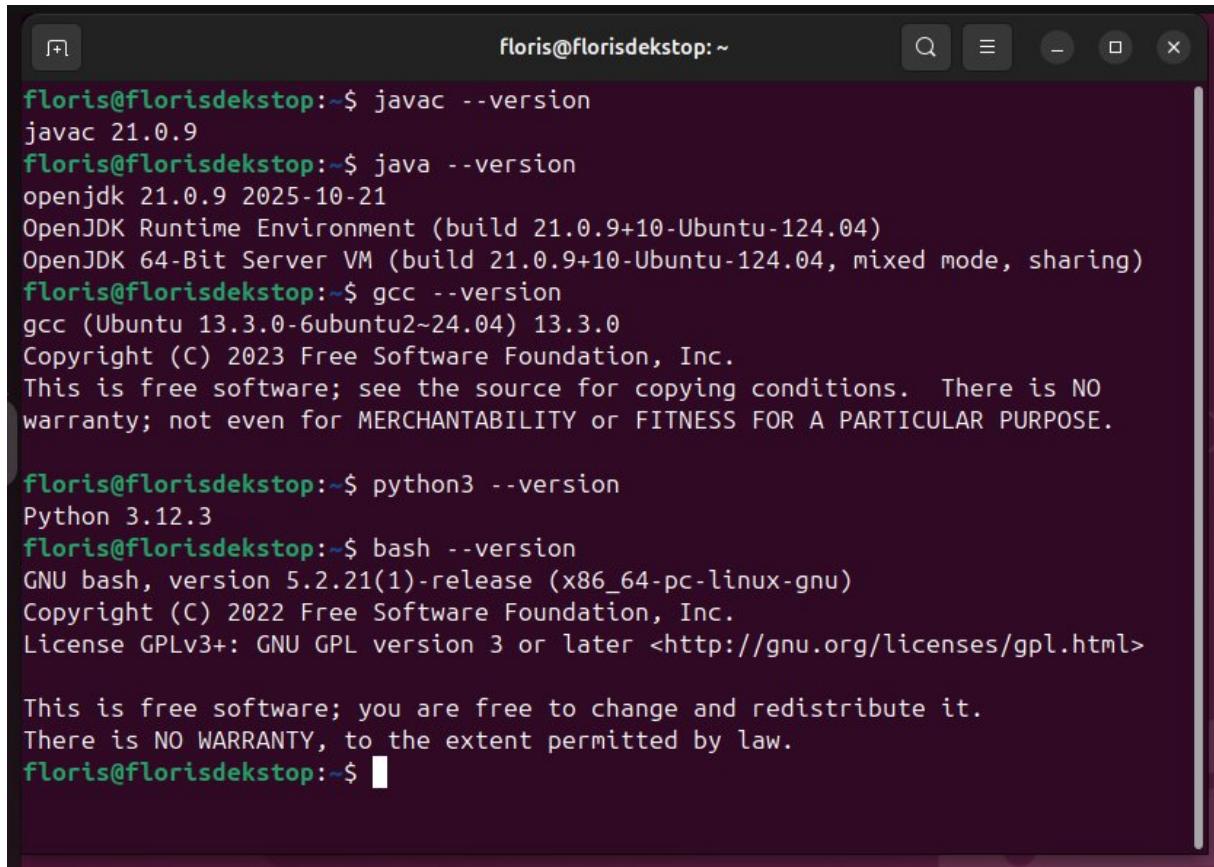
javac --version

java --version

gcc --version

python3 --version

bash --version



A screenshot of a terminal window titled "floris@florisdektop:~". The window contains the following text output:

```
floris@florisdektop:~$ javac --version
javac 21.0.9
floris@florisdektop:~$ 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)
floris@florisdektop:~$ 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.

floris@florisdektop:~$ python3 --version
Python 3.12.3
floris@florisdektop:~$ 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.
floris@florisdektop:~$
```

**JavaC: javac 21.0.9**

**Java: openjdk 21.0.9**

**GCC: 13.3.0**

**Python: 3.12.3**

**Bash: 5.2.21(1)**

### **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 en Fibonacci.java

**Which source code files are compiled to byte code?**

fib.c

**Which source code files are interpreted by an interpreter?**

Fibonacci.java

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

fib.py, fib.sh, The fastest is fib.c

**How do I run a Java program?**

1. Compile: javac Fibonacci.java
2. Run: java Fibonacci

**How do I run a Python program?**

1. python3 fib.py

How do I run a C program?

1. Compile: gcc fib.c -o fib
2. Run: ./fib

How do I run a Bash script?

1. bash fib.sh

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

C: Yes, If you run the gcc.fib it creates a default file named a.out. or a.exe on windows.

Java: Yes, The compiler makes a file with the extension .class

Python: No, You don't have to compile python, it will run it.

Bash: No bash does not compile.

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?

```
floris@florisdektop:~/Documents/code$ gcc fib.c -o fib
floris@florisdektop:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
floris@florisdektop:~/Documents/code$
```

```
floris@florisdektop:~/Documents/code$ javac Fibonacci.java
floris@florisdektop:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.37 milliseconds
floris@florisdektop:~/Documents/code$
```

```
floris@florisdektop:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.54 milliseconds
floris@florisdektop:~/Documents/code$
```

```
floris@florisdektop:~/Documents/code$ chmod +x fib.sh
floris@florisdektop:~/Documents/code$ ./fib.sh

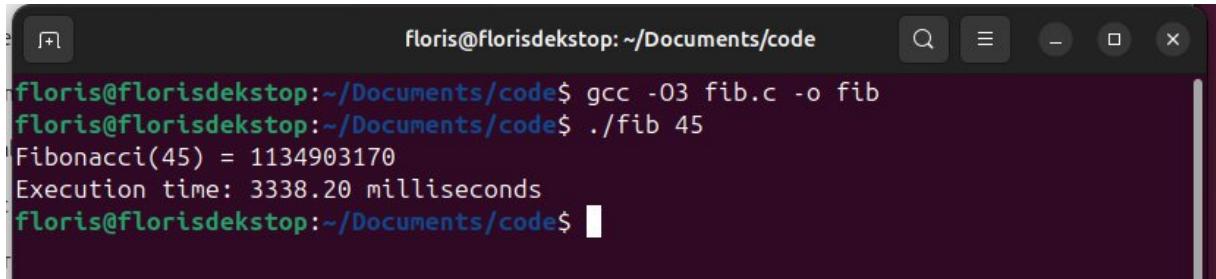
Fibonacci(18) = 2584
Excution time 12732 milliseconds
floris@florisdektop:~/Documents/code$
```

The first one (C) is the fastest.

#### 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 screenshot of a terminal window titled "floris@florisdekstop: ~/Documents/code". The terminal shows the following command sequence:

```
floris@florisdekstop:~/Documents/code$ gcc -O3 fib.c -o fib
floris@florisdekstop:~/Documents/code$ ./fib 45
Fibonacci(45) = 1134903170
Execution time: 3338.20 milliseconds
floris@florisdekstop:~/Documents/code$
```

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

```
=====
1. COMPILING SOURCE CODES
=====
[*] Compiling C (fib.c) with -O3...
[*] Compiling Java (Fibonacci.java)...

=====
2. RUNNING PERFORMANCE TEST (N=40)
=====
--- C Executable ---
Fibonacci(40) = 102334155
Execution time: 234.35 milliseconds

--- Java Bytecode ---
Fibonacci(40) = 102334155
Execution time: 668.18 milliseconds

--- Python Interpreter ---
Fibonacci(40) = 102334155
Execution time: 27276.00 milliseconds

--- Bash Script (Using N=15 for safety) ---
Fibonacci(15) = 610
Excution time 3017 milliseconds
```

other.

#### 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 a debugger window with the following sections:

- Top Bar:** open, Run, 250, Step, Reset.
- Assembly View:**

```
1 Main:  
2     mov r1, #2  
3     mov r2, #4  
4     mov r0, #1  
5  
6 Loop:  
7     cmp r2, #0  
8     beq End  
9     mul r0, r0, r1  
10    subs r2, r2, #1  
11    b Loop  
12  
13 End:  
14    bx lr
```
- Registers View:**

Register	Value
R0	2
R1	2
R2	3
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
- Memory Dump View:** Shows memory starting at address 0x00010000 with values corresponding to the assembly code.

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