

# Template Week 4 – Software

Student number: 581429

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim debugger interface. The assembly code in the left pane is:

```
1 Main:
2     mov r2, #5
3     mov r3, r2
4 Loop:
5     sub r3, r3, #1
6     mul r2, r2, r3
7     cmp r3, #1
8     bne End
9     b Loop
10 End:
```

The register values in the top right are:

Register	Value
R0	0
R1	0
R2	78
R3	1
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0

The memory dump in the bottom right shows the stack starting at address 0x00010000. The first few bytes are 05 20 A0 E3 02 30 A0 E1 01 30 43 E2 52 03 02 E0, followed by a sequence of zeros.

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

gcc --version

python3 --version

bash --version

```
volodymyr@volodymyr-Virtual-Platform:~$ javac --version
javac 21.0.8
volodymyr@volodymyr-Virtual-Platform:~$ java --version
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, sha
ring)
volodymyr@volodymyr-Virtual-Platform:~$ 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.

volodymyr@volodymyr-Virtual-Platform:~$ python3 --version
Python 3.12.3
volodymyr@volodymyr-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.
volodymyr@volodymyr-Virtual-Platform:~$
```

### **Assignment 4.3: Compile**

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

**fib.c, 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, 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?

**To run a Java program, you first compile the source file using the javac command. This turns the human-readable Java code into Java bytecode. After compilation, you run the program using the java command. So the process has two steps: first you compile the .java file, then you run the resulting class file.**

How do I run a Python program?

**You simply execute it using the Python interpreter. You type "python3 filename.py" in the terminal, where filename.py is the name of your Python script. The interpreter reads the file and runs it directly.**

How do I run a C program?

**To run a C program, you must first compile it using a C compiler such as gcc. The compiler turns the C source code into an executable program. After it is compiled, you can run the executable.**

**Example: First type "gcc fib.c -o fib" to compile the file named fib.c into an executable named fib. Then type "./fib" to run the program.**

How do I run a Bash script?

**You can run it directly using the Bash interpreter. One way is to type "bash scriptname.sh" in the terminal. Another way is to first make the script executable by running "chmod +x scriptname.sh" and then run it using "./scriptname.sh".**

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

**When you compile the C file (fib.c), a new executable file is created, for example named fib or a.out, depending on how you compiled it.**

**When you compile the Java file (Fibonacci.java), a new file named Fibonacci.class is created. This is the Java bytecode file.**

**The Python and Bash files are not compiled, so no new file is created for those.**

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?

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code". The terminal output is as follows:

```
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ javac Fibonacci.java
error: file not found: Fibonacci.java
Usage: javac <options> <source files>
      use --help for a list of possible options
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ cd ~/Desktop/code
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ javac Fibonacci.java
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.52 milliseconds
```

The image displays two screenshots of a Linux desktop environment, likely Ubuntu, running on a VMware host. Both screenshots show a desktop interface with a file manager window open and a terminal window in the foreground.

**Top Screenshot:**

- Desktop Environment:** Unity interface.
- File Manager:** Shows files in the directory `/home/volodymyr/Desktop/code`. Files include `fib`, `fib.c`, `fib.py`, `fib.sh`, `Fibonacci.class`, and `Fibonacci.java`.
- Terminal Window:** Running on a VMware Virtual Platform.

```
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ javac Fibonacci.java
error: file not found: Fibonacci.java
Usage: javac <options> <source files>
use --help for a list of possible options
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ cd ~/Desktop/code
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ javac Fibonacci.java
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.52 milliseconds
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ gcc fib.c -o fib
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$
```

**Bottom Screenshot:**

- Desktop Environment:** Unity interface.
- File Manager:** Shows files in the directory `/home/volodymyr/Desktop/code`. Files include `fib`, `fib.c`, `fib.py`, `fib.sh`, `Fibonacci.class`, and `Fibonacci.java`.
- Terminal Window:** Running on a VMware Virtual Platform.

```
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ javac Fibonacci.java
error: file not found: Fibonacci.java
Usage: javac <options> <source files>
use --help for a list of possible options
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ cd ~/Desktop/code
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ javac Fibonacci.java
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.52 milliseconds
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ gcc fib.c -o fib
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.23 milliseconds
volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code$
```

The screenshot shows a Linux desktop environment with a dark theme. A terminal window is open in the foreground, displaying the results of a benchmark test for calculating the 18th Fibonacci number. The terminal output is as follows:

```
volodymyr@volodymyr-VMware-Virtual-Platform:~/Desktop/code$ javac Fibonacci.java
volodymyr@volodymyr-VMware-Virtual-Platform:~/Desktop/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.52 milliseconds
volodymyr@volodymyr-VMware-Virtual-Platform:~/Desktop/code$ gcc fib.c -o fib
volodymyr@volodymyr-VMware-Virtual-Platform:~/Desktop/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
volodymyr@volodymyr-VMware-Virtual-Platform:~/Desktop/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.23 milliseconds
volodymyr@volodymyr-VMware-Virtual-Platform:~/Desktop/code$ bash fib.sh
Fibonacci(18) = 2584
Excution time 4288 milliseconds
volodymyr@volodymyr-VMware-Virtual-Platform:~/Desktop/code$
```

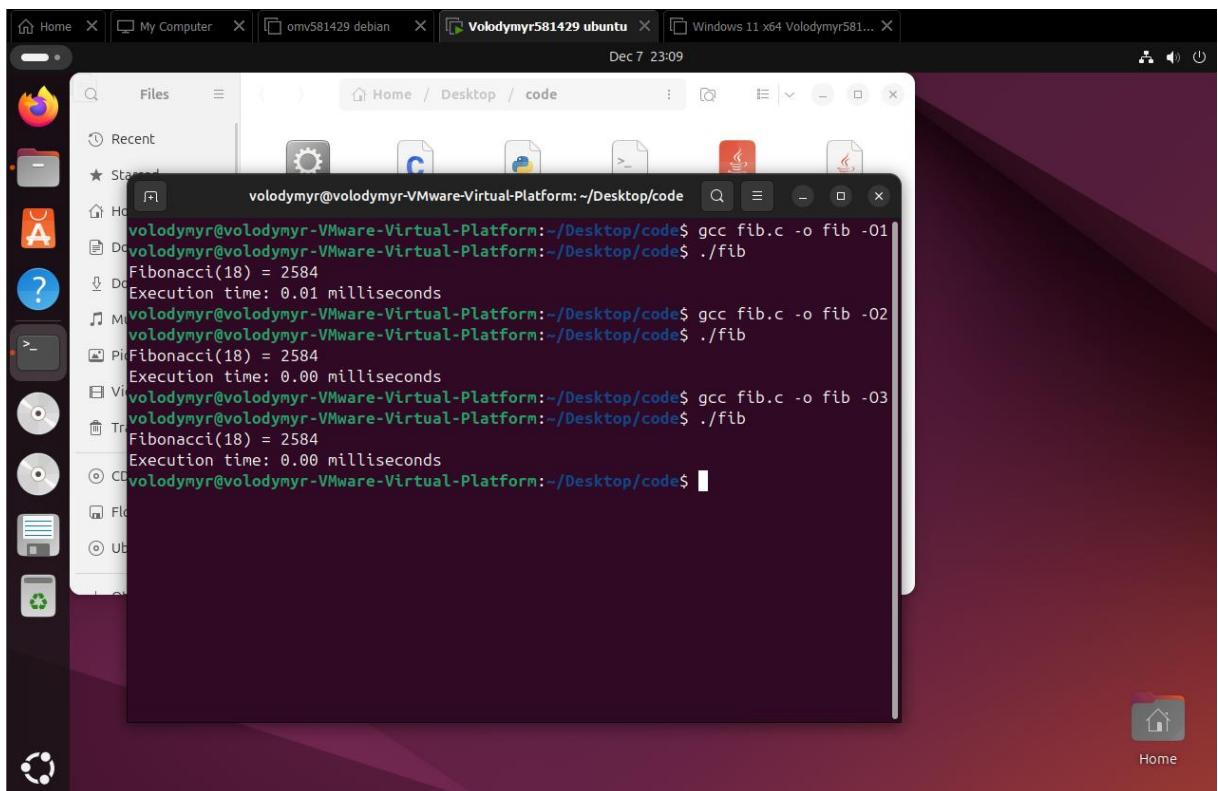
The terminal window is titled "volodymyr@volodymyr-VMware-Virtual-Platform: ~/Desktop/code". The desktop background is a purple gradient. A file manager window is visible in the background, showing files like "fib", "fib.c", "fib.py", "fib.sh", "Fibonacci.class", and "Fibonacci.java".

As can be seen on these screenshots, the fib.c file performs the fastest.

#### Assignment 4.4: Optimize

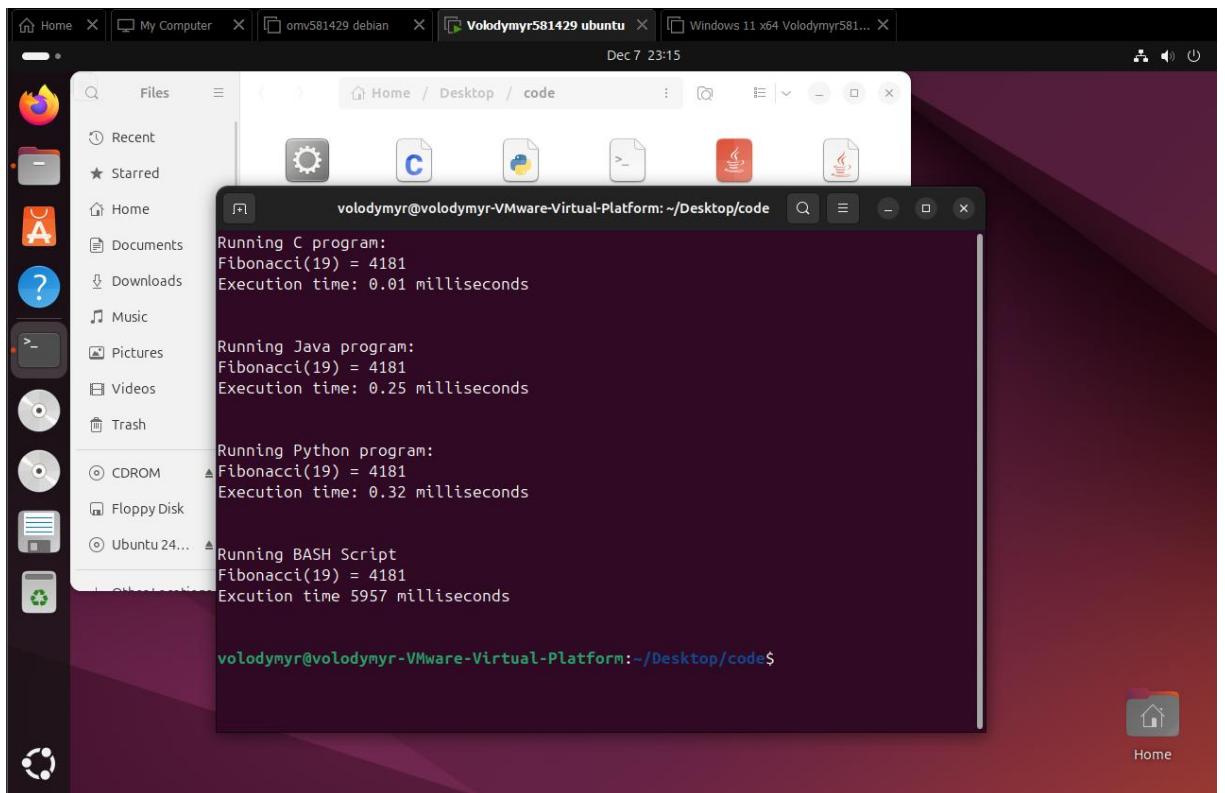
Take relevant screenshots of the following commands:

- 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.
- 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?
- 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.



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "volodymyr@volodymyr-Virtual-Platform: ~/Desktop/code". The terminal content shows the execution of a script to calculate Fibonacci numbers:

```
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ gcc fib.c -o fib -O1
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ gcc fib.c -o fib -O2
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ gcc fib.c -o fib -O3
volodymyr@volodymyr-Virtual-Platform:~/Desktop/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds
volodymyr@volodymyr-Virtual-Platform:~/Desktop/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 assembly debugger interface. At the top, there's a navigation bar with icons for back, forward, search, and file operations. Below it is a toolbar with buttons for 'Open', 'Run', 'Step', and 'Reset'. The main area has tabs for 'Registers' and 'Memory'. The 'Registers' tab displays the following register 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
R11	0
R12	0

The 'Memory' tab shows a dump of memory starting at address 0x000010000. The dump consists of 16 bytes per row, with the first byte being the address. The data is mostly zeros, with some non-zero values appearing in the higher memory addresses.

Below the memory dump, there's a status bar with the text "© 2017".

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