

Template Week 4 – Software

Student number: 589020

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim interface. On the left, there are buttons for 'Open', 'Run' (which is highlighted), '250', 'Step', and 'Reset'. Below these are the assembly instructions:

```
1 Main:  
2     mov r2, #5  
3     mov r1, #1  
4  
5 Loop:  
6     cmp r2, #1  
7     beq End  
8     mul r1, r1, r2  
9     sub r2, r2, #1  
10    b Loop  
11  
12 End:
```

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

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

Below the register table, memory dump addresses 0x00010230, 0x00010240, and 0x00010250 are shown with their corresponding hex values.

Result: 78 in hex = 120 in decimal.

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

gcc --version

python3 --version

bash --version

```
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ javac --version
javac 21.0.9
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ 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)
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ 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.

lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ python3 --version
Python 3.12.3
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ 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.
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ █
```

Assignment 4.3: Compile

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

The C file and the java file.

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

The C file which is compiled by gcc into a executable.

Which source code files are compiled to byte code?

The java file which is compiled into a .class file.

Which source code files are interpreted by an interpreter?

Fib.py. Interpreted by the Python interpreter.

Fib.sh. Interpreted by the Bash shell.

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

Most likely the C file because it compiles directly to machine code and runs natively without any virtual machine or interpreter.

How do I run a Java program?

In terminal type *javac filename.java* this creates a file named *filename.class*

Then in terminal type *java filename*

How do I run a Python program?

In terminal type *python3 filename.py*

How do I run a C program?

In terminal type *gcc -o filename filename.c*

Then type in terminal *./filename* to run the program.

How do I run a Bash script?

In terminal type *bash filename.sh*

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

Fibonacci.java -> Fibonacci.class

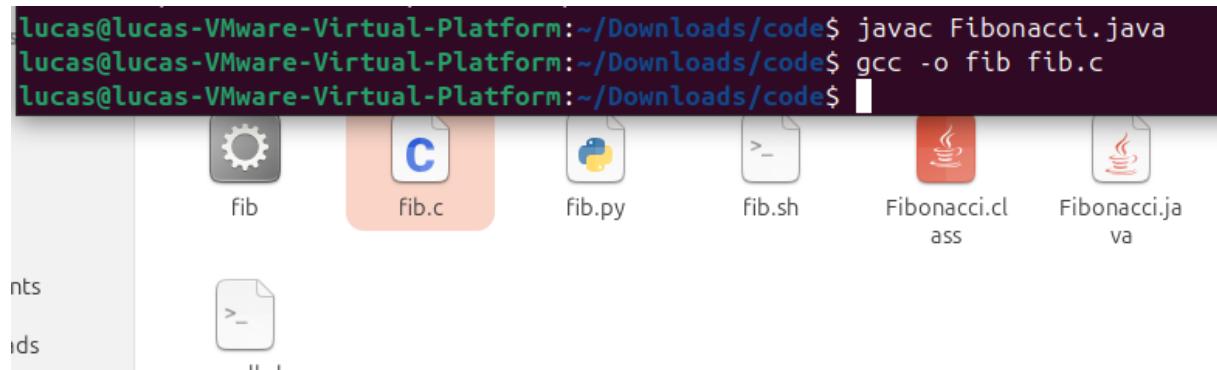
Fib.c -> fib

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?

Compiling java and c file into an executable:

```
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ gcc -o fib fib.c
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$
```



The screenshot shows a desktop environment with several icons on the desktop. From left to right, they are: a gear icon labeled 'fib', a document icon with a 'C' labeled 'fib.c' (which is highlighted in orange), a Python logo icon labeled 'fib.py', a terminal icon labeled 'fib.sh', a Java class icon labeled 'Fibonacci.class', and a Java source code icon labeled 'Fibonacci.java'. To the left of the desktop icons, there is a vertical window titled 'Downloads' containing a list of files.

Running them:

```
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.22 milliseconds
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.34 milliseconds
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ bash fib.sh
Fibonacci(18) = 2584
Execution time: 5674 milliseconds
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$
```

As you can see the C file is by far the fastest with 0.02 miliseconds. I don't know why the bash file takes more than 5 seconds.

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.

You can pass -O1, -O2, -O3 and -Ofast to optimize for execution speed. I have tried -O3 with this command: `gcc -O3 fib.c -o fib`. And it had an execution time of 0.01 milliseconds!

- 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?

```
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 fib.c -o fib
lucas@lucas-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
```

Yes it now is 0.01 milliseconds instead of 0.02 milliseconds.

- 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.

```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds
```

```
Running Java program:
Fibonacci(19) = 4181
Execution time: 0.36 milliseconds
```

```
Running Python program:
Fibonacci(19) = 4181
Execution time: 0.63 milliseconds
```

```
Running BASH Script
Fibonacci(19) = 4181
Execution time: 15612 milliseconds
```

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 interface with the following components:

- Control Buttons:** Open, Run, 250, Step, Reset.
- Assembly Code:** A list of assembly instructions numbered 1 to 13.

```
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    sub r2, r2, #1  
11    b Loop  
12  
13 End:
```
- Registers:** A table showing 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
- Memory Dump:** A table showing memory addresses and their values.

0x00010230:	00 00
0x00010240:	00 00
0x00010250:	00 00

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