Template Week 4 – Software

Student number:

578848

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

```
1  Main:
2      mov r2, #5
3      mov r1, #1
4
5  Loop:
6      mul r1, r1, r2
7      sub r2, r2, #1
8      cmp r1, #120
9      beq Exit
10     b Loop
11
12 Exit:
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

```
andrii@andrii-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
```

java --version

```
andrii@andrii-VMware-Virtual-Platform:~$ 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

```
andrii@andrii-VMware-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.
```

python3 --version

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

bash --version

```
andrii@andrii-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?

1. **.c must first be compiled into machine code**
2. **.java must be compiled into bytecode using javac**

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

1. **.c**

Which source code files are compiled to byte code?

1. **.java**

Which source code files are interpreted by an interpreter?

1. **.py**
2. **.sh**

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?

Path is to be replaced with an actual path

java *path*

How do I run a Python program?

python3 *path*

How do I run a C program?

gcc *path -o program*

./out_file_path

How do I run a Bash script?

bash *path*

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

.java will create a ClassName.class

.c will create a machine code file, which is either file.out (Linux) or file.exe (Win)

.py will not, except .pyc cache bytecode files

Take relevant screenshots of the following commands:

- Compile the source files where necessary

```
andrii@andrii-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
andrii@andrii-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
andrii@andrii-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c
andrii@andrii-VMware-Virtual-Platform:~/Downloads/code$ ls
a.out  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
```

- Make them executable

```
andrii@andrii-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x fib.sh
[sudo] password for andrii:
andrii@andrii-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x runall.sh
```

- Run them

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


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.24 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.46 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 7330 milliseconds
```

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

.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.

The command:
gcc -O3 -march=native -flto -funroll-loops fib.c -o fib
Takes it as far as possible

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

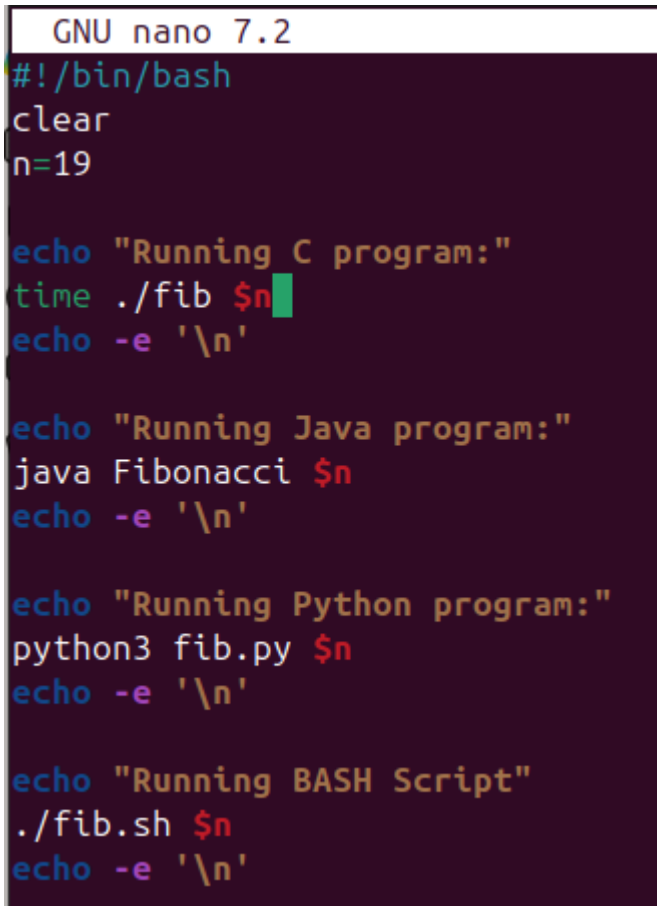
```
andrii@andrii-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 -march=native -f
lto -funroll-loops fib.c -o fib
```

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

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

real    0m0.001s
user    0m0.001s
sys     0m0.000s


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.23 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.44 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 7254 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.

I did it like this, and used time for more precise time metrics.



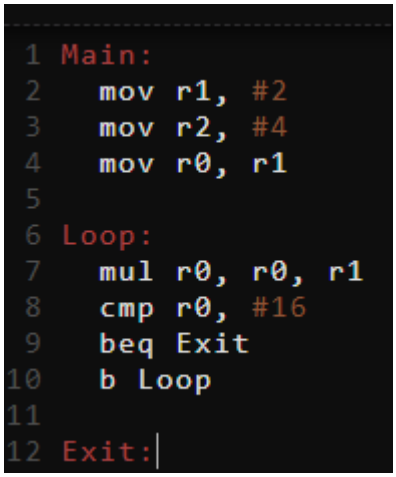
Assignment 4.5: More ARM Assembly

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.



```
1  Main:
2    mov r1, #2
3    mov r2, #4
4    mov r0, r1
5
6  Loop:
7    mul r0, r0, r1
8    cmp r0, #16
9    beq Exit
10   b Loop
11
12 Exit:
```

Ready? Save this file and export it as a pdf file with the name: **week4.pdf**