

Template Week 4 – Software

Student number: 589845

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a debugger interface with the following details:

- Top menu: Open, Run (highlighted), 250, Step, Reset.
- Assembly code:

```
1 Main:
2     mov r2, #5
3     mov r1, r2
4 Loop:
5     sub r2, #1
6     mul r1, r2, r1
7
8     cmp r2, #1
9     beq End
10    b Loop
11
12 End:
13
```
- Registers table:

Register	Value
R0	0
R1	78
R2	1
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
sp	10000
- Memory dump:

```
0x00010000: 05 20 A0 E3 02 10 A0 E1 01 20 42 E2 92 01 01 E0
0x00010010: 01 52 E3 00 00 0A FA FF FF EA 00 00 00 00 00 00 R
0x00010020:
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

```
maickel@Maickel:~$ javac --version
javac 21.0.9
```

java --version

```
maickel@Maickel:~$ 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

```
maickel@Maickel:~$ 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

```
maickel@Maickel:~$ python3 --version
Python 3.12.3
```

bash --version

```
maickel@Maickel:~$ 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?

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?

Compile:

javac Fibonacci.java

Run:

java Fibonacci

How do I run a Python program?

python3 fib.py

How do I run a C program?

Compile:

gcc fib.c -o fib

Run:

./fib

How do I run a Bash script?

bash fib.sh

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

fib and Fibonacci.class, So the C and Java will create a new file.

Take relevant screenshots of the following commands:

- Compile the source files where necessary

```
maickel@Maickel:~/Downloads/code$ javac Fibonacci.java  
maickel@Maickel:~/Downloads/code$ gcc fib.c -o fib
```

- Make them executable

```
maickel@Maickel:~/Downloads/code$ sudo chmod a+x fib  
maickel@Maickel:~/Downloads/code$ sudo chmod a+x Fibonacci.class  
maickel@Maickel:~/Downloads/code$ sudo chmod a+x fib.py  
maickel@Maickel:~/Downloads/code$ sudo chmod a+x fib.sh
```

- Run them

```
maickel@Maickel:~/Downloads/code$ sudo ./fib  
Fibonacci(18) = 2584  
Execution time: 0.04 milliseconds  
maickel@Maickel:~/Downloads/code$ sudo ./fib.sh  
Fibonacci(18) = 2584  
Excution time 7517 milliseconds  
maickel@Maickel:~/Downloads/code$ java Fibonacci  
Fibonacci(18) = 2584  
Execution time: 0.30 milliseconds  
maickel@Maickel:~/Downloads/code$ python3 fib.py  
Fibonacci(18) = 2584  
Execution time: 0.47 milliseconds
```

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

The C file.

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.
-O[Level] (Level: 00 > 03)

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

Gcc -O3 -o fib fib.c

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

```
maickel@Maickel:~/Downloads/code$ sudo ./fib
Fibonacci(18) = 2584
Execution time: 0.04 milliseconds
maickel@Maickel:~/Downloads/code$ gcc -O3 -o fib fib.c
maickel@Maickel:~/Downloads/code$ sudo ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
maickel@Maickel:~/Downloads/code$
```

- 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: 0.00 milliseconds
```

```
Running Java program:  
Fibonacci(19) = 4181  
Execution time: 0.28 milliseconds
```

```
Running Python program:  
Fibonacci(19) = 4181  
Execution time: 0.32 milliseconds
```

```
Running BASH Script  
Fibonacci(19) = 4181  
Excution time 5978 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 the QEMU debugger interface. The top bar includes buttons for Open, Run (highlighted), Step, and Reset. The assembly code window displays the following sequence:

```
1 Main:  
2     mov r1, #2  
3     mov r3, r1  
4     mov r2, #4  
5  
6 Loop:  
7     mul r3, r3, r1  
8     sub r2, #1  
9     cmp r2, #1  
10    bge End  
11    b Loop  
12  
13 End:  
14     mov r0, r3
```

The Registers window shows the following values:

Register	Value
R0	10
R1	2
R2	1
R3	10
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
sp	10000

The Memory dump window shows memory starting at address 0x0000100000:

Address	Value
0x0000100000	02 10 A0 E3 01 30 A0 E1 04 20 A0 E3 93 01 03 E0 ... 0
0x0000100004	01 20 42 E2 01 00 52 E3 00 00 00 0A FA FF FF EA B ... R
0x0000100008	E3 A0 E1
0x000010000C	00 00 00 00

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