

Template Week 4 – Software

Student number: 547518

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a debugger interface with the following details:

- Toolbar:** Open, Run, 250, Step, Reset.
- Assembly Code:**

```
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 r2, #1
9     beq End
10    b Loop
11
12 End:
13 |
```
- Registers:** A table showing register values:

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:** Hex dump of memory starting at address 0x00010000.

Address	Value
0x00010000	05 20 F0
0x00010010	01 00 E5
0x00010020	00 00 00
0x00010030	00 00 00
0x00010040	00 00 00
0x00010050	00 00 00
0x00010060	00 00 00
0x00010070	00 00 00

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac –version

```
Marco547518@Ubuntu-VM:~$ javac --version
javac 17.0.17
```

java –version

```
Marco547518@Ubuntu-VM:~$ java --version
openjdk 17.0.17 2025-10-21
OpenJDK Runtime Environment (build 17.0.17+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 17.0.17+10-Ubuntu-124.04, mixed mode, sharing)
```

gcc –version

```
Marco547518@Ubuntu-VM:~$ 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

```
Marco547518@Ubuntu-VM:~$ python3 --version
Python 3.12.3
```

bash –version

```
Marco547518@Ubuntu-VM:~$ 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? **Fibonacci.java and fib.c**

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 and 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? **Run javac Fibonacci.java to compile it. Thus creating Fibonacci.class and then run java Fibonacci to run the program**

How do I run a Python program? **Just run python3 fib.py**

How do I run a C program? **First compile it with gcc like this: gcc fib.c -o fib Then it creates fib then make it an executable with: chmod +x fib then run ./fib**

How do I run a Bash script? **Make the script executable with sudo chmod a+x fib.sh then run sudo ./fib.sh**

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

Yes fib.c will create fib and Fibonacci.java will create Fibonacci.class

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?

```
Marco547518@Ubuntu-VM:~/Downloads/code$ javac Fibonacci.java
Marco547518@Ubuntu-VM:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.33 milliseconds
```

Example of java compilation for Fibonacci.java

```
Marco547518@Ubuntu-VM:~/Downloads/code$ gcc fib.c -o fib
Marco547518@Ubuntu-VM:~/Downloads/code$ chmod +x fib
Marco547518@Ubuntu-VM:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
```

Example of C compilation for fib.c . C has the fastest execution time out of all the programs

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. There are flags like -O0 which is on my default that uses no optimizations and it goes until -O3 but with research it appears that -O2 is the best for performance as -O3 increases the file size

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

Ran the command `gcc -O2 fib.c -o fib_opt`

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

```
Marco547518@Ubuntu-VM:~/Downloads/code$ gcc -O2 fib.c -o fib_opt
Marco547518@Ubuntu-VM:~/Downloads/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.00 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: 0.03 milliseconds
```

```
Running Java program:
Fibonacci(19) = 4181
Execution time: 0.33 milliseconds
```

```
Running Python program:
Fibonacci(19) = 4181
Execution time: 1.52 milliseconds
```

```
Running BASH Script
Fibonacci(19) = 4181
Excution time 10328 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:

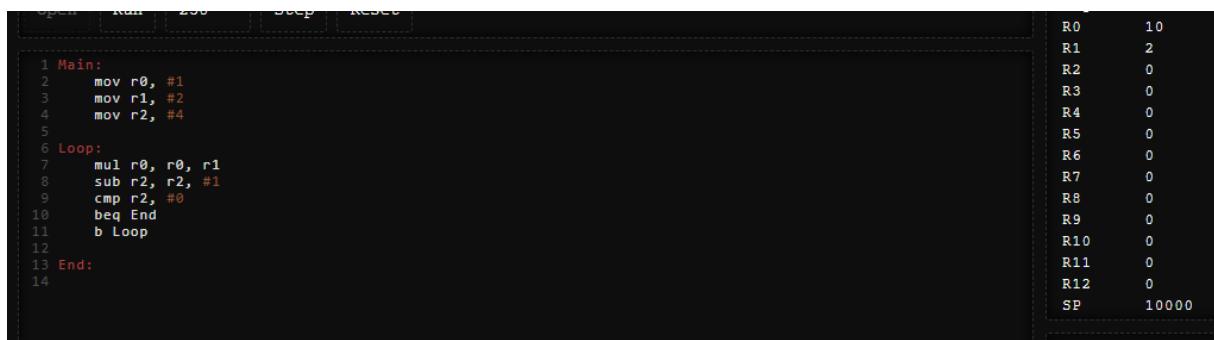
```
mov r1, #2  
mov r2, #4
```

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 two main sections: assembly code on the left and register values on the right.

Assembly Code (Left):

```
1 Main:  
2     mov r0, #1  
3     mov r1, #2  
4     mov r2, #4  
5  
6 Loop:  
7     mul r0, r0, r1  
8     sub r2, r2, #1  
9     cmp r2, #0  
10    beq End  
11    b Loop  
12  
13 End:  
14
```

Registers (Right):

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
SP	10000

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