

# Template Week 4 – Software

Student number: 580521

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a debugger interface with the following details:

- Registers:** R0=0, R1=78, R2=0, R3=0, R4=0, R5=0, R6=0, R7=0, R8=0, R9=0, R10=0.
- Memory Dump:** A hex dump of memory starting at address 0x00001000, showing the assembly code and its binary representation.
- Assembly Code:**

```
1 Main:
2     mov r0, #5
3     mov r1, #1
4
5 Loop:
6     mul r1, r1, r0
7     subs r0, r0, #1
8     bne Loop
9
10 END:
11
```

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

--version python3

--version

bash --version

The terminal window displays the following command outputs:

- javac --version**: javac 21.0.9
- 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)
- python3 --version**: Python 3.12.3
- 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.
- 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?

- **Java and C must be compiled**

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

- C

Which source code files are compiled to byte code?

- Java

Which source code files are interpreted by an interpreter?

- **Python and Bash**

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

- C

```
constantin@580521:~/Desktop/folder$ javac Fibonacci.java
constantin@580521:~/Desktop/folder$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.29 milliseconds
constantin@580521:~/Desktop/folder$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.40 milliseconds
constantin@580521:~/Desktop/folder$ gcc fib.c -o fib

constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
constantin@580521:~/Desktop/folder$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.shh
constantin@580521:~/Desktop/folder$ chmod +x fib.shh
constantin@580521:~/Desktop/folder$ ./fib.shh
Fibonacci(18) = 2584
Excution time 6945 milliseconds
constantin@580521:~/Desktop/folder$
```

How do I run a Java program?

- javac Fibonacci.java
- java Fibonacci

How do I run a Python program?

- python3 fib.py

How do I run a C program?

- gcc fib.c -o fib
- ./fib

How do I run a Bash script?

- chmod +x fib.shh
- ./fib.shh

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?

```
constantin@580521:~/Desktop/folder$ javac Fibonacci.java
Fibonacci.java:1: error: class Fibonacci is public, should be declared in a file
named Fibonacci.java
public class Fibonacci {
^
1 error
constantin@580521:~/Desktop/folder$ mv Fibonacci.java Fibonacci.java
constantin@580521:~/Desktop/folder$ javac Fibonacci.java
constantin@580521:~/Desktop/folder$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.29 milliseconds
constantin@580521:~/Desktop/folder$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.40 milliseconds
constantin@580521:~/Desktop/folder$ gcc fib.c -o fib

constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
constantin@580521:~/Desktop/folder$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.shh
constantin@580521:~/Desktop/folder$ chmod +x fib.shh
constantin@580521:~/Desktop/folder$ ./fib.shh
Fibonacci(18) = 2584
Execution time 6945 milliseconds
constantin@580521:~/Desktop/folder$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.shh
constantin@580521:~/Desktop/folder$
```

#### 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.

- -O0 • -O1 • -O2
- -O3
- -Ofast

- 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?

```
constantin@580521:~/Desktop/folder$ gcc -O0 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
constantin@580521:~/Desktop/folder$ gcc -O1 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
constantin@580521:~/Desktop/folder$ gcc -O2 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
constantin@580521:~/Desktop/folder$ gcc -O3 fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
constantin@580521:~/Desktop/folder$ gcc -Ofast fib.c -o fib
constantin@580521:~/Desktop/folder$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
constantin@580521:~/Desktop/folder$ █
```

- 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.

#### 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
    mov r0, #1
Loop:
    cmp r2, #0
    beq End      mul r0,
    r1, r0      subs r2,
    r2, #1
    b Loop

```

End:

b End

Complete the code. See the PowerPoint slides of week 4.

Screenshot of the completed code here.

The screenshot shows a debugger window with the following components:

- Top Bar:** Open, Run, 250, Step, Reset.
- Left Panel (Assembly View):**

```

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, r1, r0
10  subs r2, r2, #1
11  b Loop
12
13 End:
14 b End
15

```
- Right Panel (Registers View):**

Register	Value
R0	10
R1	2
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
- Bottom Panel (Memory View):** A hex dump of memory starting at address 0x000010000. The first few lines show:

```

0x000010000: 02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 00 00 52 E3
0x000010010: 02 00 00 0A 91 00 00 E0 01 20 52 E2 FA FF FF EA
0x000010020: FF FF FF EA 00 00 00 00 00 00 00 00 00 00 00 00
0x000010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000100F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

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