

Template Week 4 – Software

Student number: 580606

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim debugger interface. On the left, the assembly code for a factorial program is displayed:

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

On the right, the register values and memory dump are shown:

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

Memory dump (hex dump):

```
0x00010000: 05 20 A0 E3 01 10 A0 E3 FF FF EA 91 02 01 E0 . .
0x00010010: 01 20 42 E2 01 00 52 E3 00 00 00 00 0A FA FF EA . .
0x00010020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x00010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x00010040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x00010050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x00010060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x00010070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x00010080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x00010090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x000100A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
0x000100B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . .
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

gcc --version

python3 --version

bash --version

```
sandra@sandra-VMware20-1:~$ javac --version
javac 21.0.9
sandra@sandra-VMware20-1:~$ 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)
sandra@sandra-VMware20-1:~$ 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.

sandra@sandra-VMware20-1:~$ python3 --version
Python 3.12.3
sandra@sandra-VMware20-1:~$ bash --version
GNU bash, version 5.2.21(1)-release (aarch64-unknown-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.
sandra@sandra-VMware20-1:~$ 
```

Assignment 4.3: Compile

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

Files fib.c and Fibonacci.java have to be compiled first, because C and Java require a compiler before execution.

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

File fib.c is compiled into machine code and then is executed by the computer's processor.

Which source code files are compiled to byte code?

File Fibonacci.java is converted into bytecode before being executed by a VM.

Which source code files are interpreted by an interpreter?

File fib.py is executed by an interpreter.

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

The fastest of the codes should be the fib.c file, because C is a compiled language and the code is translated into machine code, which can then be directly executed by the processor.

How do I run a Java program?

First, you have to make sure you have jdk installed and then run the following commands in Terminal:

javac file.java

java file

The first line compiles the code and the second one executes it.

How do I run a Python program?

First, you have to make sure you have python installed and then run the following command in Terminal:

python3 file.py

How do I run a C program?

First, you have to make sure you have gcc installed and then run the following commands in Terminal:

gcc file.c -o run_file

./run_file

The first line compiles the code and the second one executes it.

How do I run a Bash script?

You have to run the following commands in Terminal:

chmod +x ./file.sh

./file.sh

The first line gives executable permission to the bash file and the second line runs it.

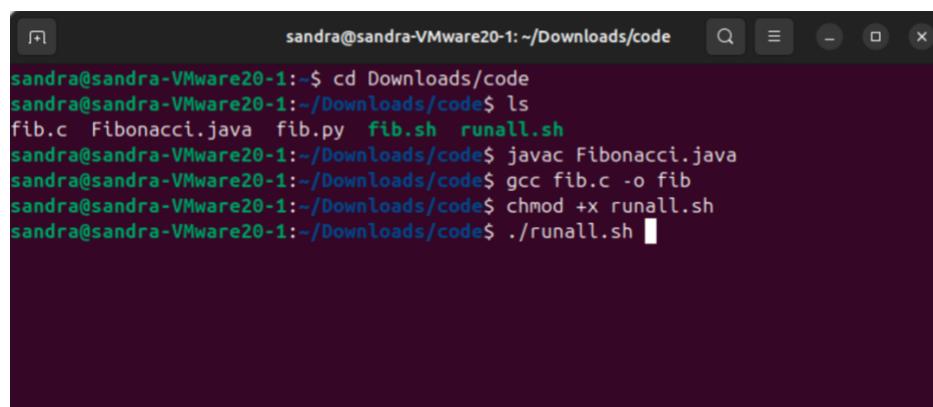
If I compile the above source code, will a new file be created? If so, which file?

Yes, an executable file will be created from the C source code file and a .class file will be created from the .java file.

Take relevant screenshots of the following commands:

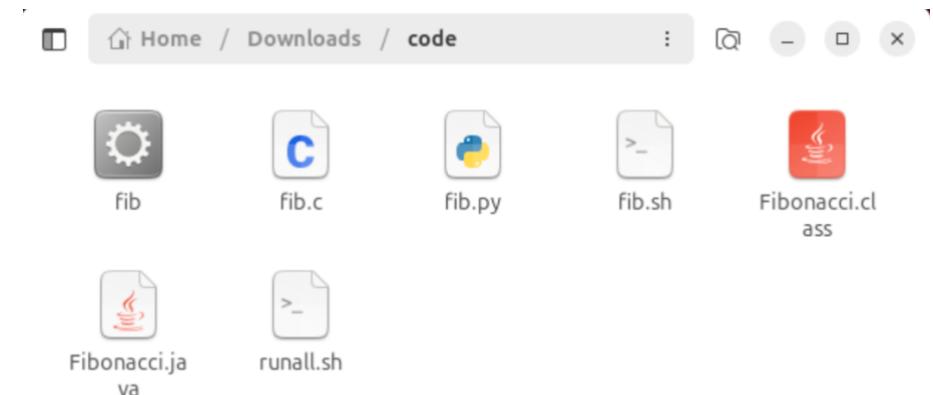
- Compile the source files where necessary
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?

Compiling the C and java code, giving executable permission to the runall bash script:



```
sandra@sandra-VMware20-1:~$ cd Downloads/code
sandra@sandra-VMware20-1:~/Downloads/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
sandra@sandra-VMware20-1:~/Downloads/code$ javac Fibonacci.java
sandra@sandra-VMware20-1:~/Downloads/code$ gcc fib.c -o fib
sandra@sandra-VMware20-1:~/Downloads/code$ chmod +x runall.sh
sandra@sandra-VMware20-1:~/Downloads/code$ ./runall.sh
```

Two new files fib and Fibonacci.class were created:



The result of executing the runall bash script:

```
sandra@sandra-VMware20-1:~/Downloads/code$ javac Fibonacci.java
sandra@sandra-VMware20-1:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.22 milliseconds
sandra@sandra-VMware20-1:~/Downloads/code$
```

```
sandra@sandra-VMware20-1:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.46 milliseconds
```

```
sandra@sandra-VMware20-1:~/Downloads/code$ gcc -o fib fib.c
sandra@sandra-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
sandra@sandra-VMware20-1:~/Downloads/code$
```

```
sandra@sandra-VMware20-1:~/Downloads/code$ chmod +x ./fib.sh
sandra@sandra-VMware20-1:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 1907 milliseconds
```

The fastest calculation was performed by the C source code.

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

```
sandra@sandra-VMware20-1:~$ cd Downloads/code
sandra@sandra-VMware20-1:~/Downloads/code$ gcc -O2 -o fib fib.c
sandra@sandra-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
sandra@sandra-VMware20-1:~/Downloads/code$ █
```

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

Yes, without the optimization parameters the code runs in 0.03 milliseconds, and with the optimization it compiles in 0.01 millisecond.

- 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: 0.05 milliseconds

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
Execution time: 3209 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 OakSim assembly debugger interface. The assembly code is as follows:

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

The register values are:

Register	Value
R0	10
R1	2
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0

The memory dump shows the following memory dump starting at address 0x000010000:

Address	Value
0x000010000	02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 90 01 00 E0
0x000010010	01 20 42 E2 00 00 52 E3 00 00 00 FA FF FF EA
0x000010020	00 00 00 00 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