

# Template Week 4 – Software

Student number:

580693

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim software interface. On the left, there is a code editor window displaying ARM assembly code. The code starts with a Main routine, initializes registers R0 through R7, and then enters a loop where it multiplies R1 by R2, decrements R2, and checks if R2 is zero. If R2 is not zero, it loops back. Once R2 reaches zero, it exits the loop. On the right, there is a register viewer showing the current values of the registers: R0=0, R1=78, R2=1, R3=0, R4=0, R5=0, R6=0, and R7=0. Below the register viewer is a memory dump window showing memory addresses from 0x00010000 to 0x00010150. The memory dump shows the assembly code and some initial values.

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

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

Address	Value
0x00010000:	05 20 A0 E3 01 10 A0 E3 82 01 01 E0 01 20 42 E2
0x00010010:	01 00 52 E5 00 00 00 00 00 00 00 00 00 00 00 00
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
0x000100C0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000100D0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000100E0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000100F0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010100:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010110:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010120:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010130:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010140:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010150:	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

### Assignment 4.3: Compile

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

The java and c files must be compiled before executing them.

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

The python and bash files are compiled into machine code and then directly executable.

Which source code files are compiled to byte code?

Java and C are compiled to byte code.

Which source code files are interpreted by an interpreter?

Python and Bash are interpreted 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 c file will do the calculation the fastest.

How do I run a Java program?

Using the javac command to first compile it, then running the java "nameOfFile" command.

How do I run a Python program?

Using the python3 command – it doesn't need to be compiled beforehand.

How do I run a C program?

Using the gcc command first to compile the code, then executing the executable file made from the compiling.

How do I run a Bash script?

Using the bash command or just ./"nameOfFile".

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

Yes, a .class file will be created for the java file and an executable file will be created for the c 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?

```
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ gcc -o fib fib.c
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ ls
fib fib.c Fibonacci.java fib.py fib.sh runall.sh
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Error: Could not find or load main class Fibonacci
Caused by: java.lang.ClassNotFoundException: Fibonacci
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci.java
Fibonacci(18) = 2584
Execution time: 0.30 milliseconds
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ python3 Fibonacci.py
python3: can't open file '/home/kliment/Downloads/code/Fibonacci.py': [Errno 2]
No such file or directory
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.47 milliseconds
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ bash fib.sh
Fibonacci(18) = 2584
Excution time 8619 milliseconds
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$
```

#### Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

- 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.

```
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 fib.c -o fib
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$
```

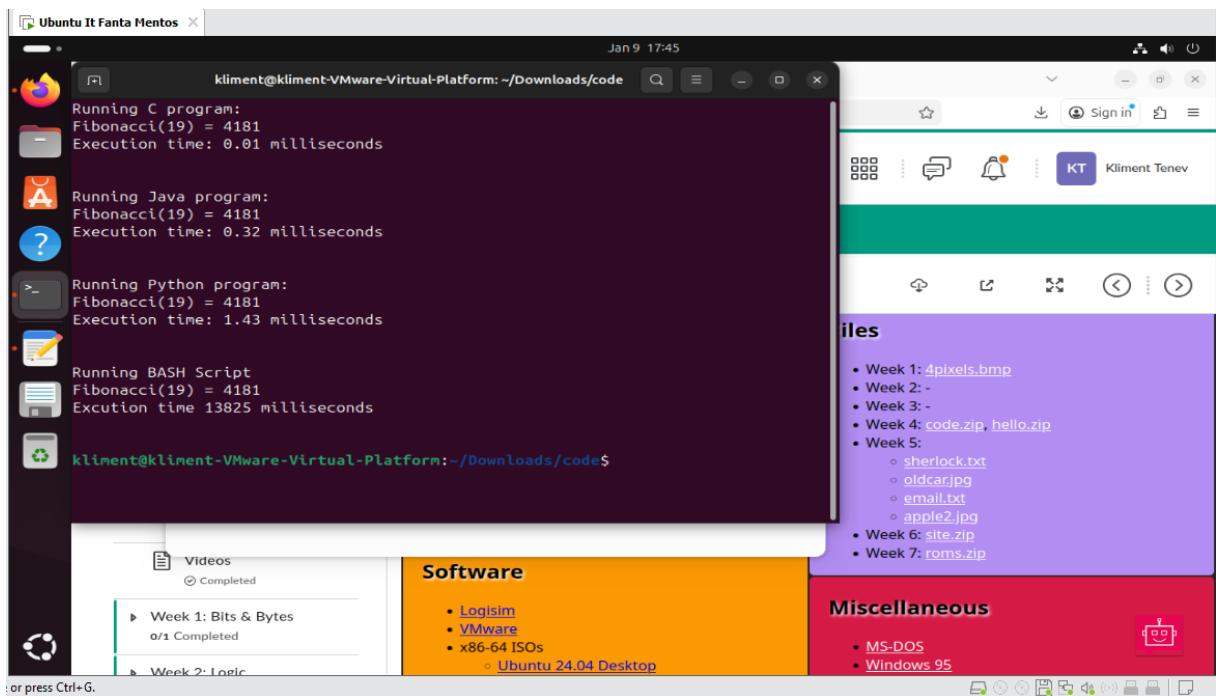
- Compile **fib.c** again with the optimization parameters

```
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
kliment@kliment-VMware-Virtual-Platform:~/Downloads/code$
```

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

Yes it went from 0.03 to 0.01 milliseconds

- 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
```

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. On the left, the assembly code is displayed:

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

On the right, the register values are listed:

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

The memory dump window at the bottom shows the memory starting at address 0x00010000. The first few bytes are 02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 00 00 52 E3, followed by several zeros. The timestamp at the bottom right is 17:37 9.1.2026 r.

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