

# Template Week 4 – Software

Student number: 571927

## 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=2, R1=4, R2=2, R3=f, R4=0, R5=0, R6=0, R7=0, R8=0, R9=0, R10=0, R11=0, R12=0, SF=10000.
- Memory Dump:** A dump of memory starting at address 0x00010000, showing hex values for each byte.
- Assembly Code:** The code is a loop that adds the value of R0 to R1, multiplies R1 by R2, subtracts R2 from R1, and moves the result to R3. The loop continues until R3 is 15.
- Buttons:** Open, Run (highlighted), Step, Reset.

Values:

r0: 2  
r1: 4  
r2: 2  
r3: f  
r4: 0

Dasm

Register	Value
R0	5
R1	10
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
SP	10000

```

1 Loop:
2   add r0, r0, #1
3   mul r1, r0, r0
4   cmp r1, #144
5   bne Exit
6   b Loop
7
8 Exit:
9

```

© 2017

Dasm

Register	Value
R0	5
R1	4
R2	3
R3	2
R4	1
R5	0
R6	0
R7	0
R8	0
R9	14
R10	3c
R11	78
R12	78
SP	10000

```

1 Loop:
2   mov r0, #5
3   mov r1, #4
4   mov r2, #3
5   mov r3, #2
6   mov r4, #1
7
8   mul r5, r1, r0
9   mul r10, r9, r2
10  mul r11, r10, r3
11  mul r12, r11, r4
12

```

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac -version

```
stijn@ubuntu-591527:~$ javac --version
Command 'javac' not found, but can be installed with:
sudo apt install openjdk-17-jdk-headless # version 17.0.17+10-1~24.04, or
sudo apt install openjdk-21-jdk-headless # version 21.0.9+10-1~24.04
sudo apt install default-jdk          # version 2:1.17-75
sudo apt install openjdk-11-jdk-headless # version 11.0.29+7-1ubuntu1~24.04
sudo apt install openjdk-25-jdk-headless # version 25.0.1+8-1~24.04
sudo apt install openjdk-8-jdk-headless # version 8u472-ga-1~24.04
sudo apt install ecj                  # version 3.32.0+eclipse4.26-2
sudo apt install openjdk-19-jdk-headless # version 19.0.2+7-4
sudo apt install openjdk-20-jdk-headless # version 20.0.2+9-1
sudo apt install openjdk-22-jdk-headless # version 22~22ea-1
stijn@ubuntu-591527:~$
```

java -version

```
stijn@ubuntu-591527:~$ java --version
Command 'java' not found, but can be installed with:
sudo apt install openjdk-17-jre-headless # version 17.0.17+10-1~24.04, or
sudo apt install openjdk-21-jre-headless # version 21.0.9+10-1~24.04
sudo apt install default-jre          # version 2:1.17-75
sudo apt install openjdk-11-jre-headless # version 11.0.29+7-1ubuntu1~24.04
sudo apt install openjdk-25-jre-headless # version 25.0.1+8-1~24.04
sudo apt install openjdk-8-jre-headless # version 8u472-ga-1~24.04
sudo apt install openjdk-19-jre-headless # version 19.0.2+7-4
sudo apt install openjdk-20-jre-headless # version 20.0.2+9-1
sudo apt install openjdk-22-jre-headless # version 22~22ea-1
stijn@ubuntu-591527:~$ S
```

gcc -version

```
stijn@ubuntu-591527:~$ 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.

stijn@ubuntu-591527:~$
```

python3 -version

```
stijn@ubuntu-591527:~$ python3 --version
Python 3.12.3
stijn@ubuntu-591527:~$
```

bash -version

```
stijn@ubuntu-591527:~$ 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.
stijn@ubuntu-591527:~$
```

### Assignment 4.3: Compile

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

Fib.c, fib.py, 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, fib.py

Which source code files are interpreted by an interpreter?

Fib.sh, runall.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?

Eerst compileren, dan daarna uitvoeren

How do I run a Python program?

Je gaat eerst in de terminal naar het pad van het bestand. Daarna run je 'python (bestandsnaam)'

How do I run a C program?

Bijna dezelfde maniera ls bij Java. Eerst compileer je de code, daarna voer je de gecompileerde code uit.

How do I run a Bash script?

Je voert hem letterlijk uit. Eventueel kan je zelfs de commando's handmatig in CMD plakken. Ik kan er niet veel meer van maken.

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

Verschilt per code dat je wilt compileren.

Als je het .c bestand compileerd, krijg je waarschijnlijk een .exe bestand.

Als je het .java bestand compileert, krijg je waarschijnlijk een (naam).class bestand. En dit is Java bytecode.

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?

```
stijn@ubuntu-591527:~/Downloads/code$ gcc fib.c -o fib-uitvoerbaar
stijn@ubuntu-591527:~/Downloads/code$ ./fib-uitvoerbaar
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
stijn@ubuntu-591527:~/Downloads/code$ 
```

```
stijn@ubuntu-591527:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.32 milliseconds
stijn@ubuntu-591527:~/Downloads/code$ 
```

```
stijn@ubuntu-591527:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 7910 milliseconds
stijn@ubuntu-591527:~/Downloads/code$ 
```

```
stijn@ubuntu-591527:~/Downloads/code$ javac Fibonacci.java
stijn@ubuntu-591527:~/Downloads/code$ java Fibonacci.java
Fibonacci(18) = 2584
Execution time: 0.53 milliseconds
stijn@ubuntu-591527:~/Downloads/code$ 
```

Runall.sh

```
stijn@ubuntu-591527:~/Downloads/code$ ./runall.sh
Running C program:
./runall.sh: line 6: ./fib: No such file or directory

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.50 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.75 milliseconds

Running BASH Script
```

#### 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.

```
stijn@ubuntu-591527:~/Downloads/code$ ./fib-uitvoerbaar
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
stijn@ubuntu-591527:~/Downloads/code$
```

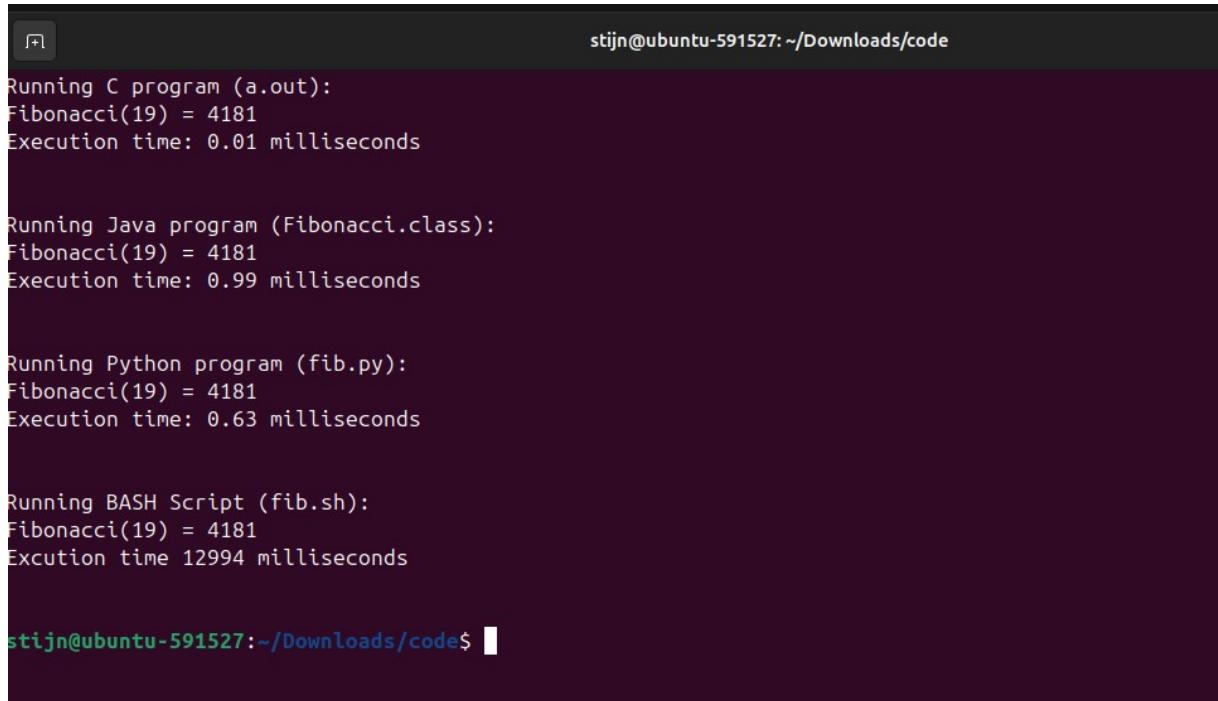
- Compile **fib.c** again with the optimization parameters

```
stijn@ubuntu-591527:~/Downloads/code$ gcc -O3 fib.c
stijn@ubuntu-591527:~/Downloads/code$ ./a.out
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
stijn@ubuntu-591527:~/Downloads/code$
```

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

Yes. 0.02 milliseconds faster than before.

- 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.



```
stijn@ubuntu-591527: ~/Downloads/code$ ./runall.sh
Running C program (a.out):
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds

Running Java program (Fibonacci.class):
Fibonacci(19) = 4181
Execution time: 0.99 milliseconds

Running Python program (fib.py):
Fibonacci(19) = 4181
Execution time: 0.63 milliseconds

Running BASH Script (fib.sh):
Fibonacci(19) = 4181
Excution time 12994 milliseconds

stijn@ubuntu-591527:~/Downloads/code$
```

## 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 OxSim assembly debugger interface. On the left, the assembly code is displayed:

```

1  MAIN:
2      ; Registers erin knallen
3      mov r0, #1
4      mov r1, #2
5      mov r2, #4
6
7  Loop:
8      ; Is loopje afgelopenen?
9      cmp r2, #0
10     beq End
11
12     ; Nu berkeenen die handel
13     mul r0, r0, r1
14
15     ; Nu naar de volgende
16     sub r2, r2, #1
17
18     b Loop
19 End:
20     bx lr

```

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

Register	Value
R0	0
R1	2
R2	4
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
SF	10000

Memory dump (0x00010000 to 0x00010FE0):

```

0x00010000: 02 30 A0 E3 04 20 A0 E3 00 00 00 0A FD FF FF EA ./
0x00010010: 1E FF 2F E1 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

```

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