

Template Week 4 – Software

Student number: 589892

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:** Hex dump of memory starting at address 0x00010000, showing the assembly code and its binary representation.

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

```
0x00010000: 05 20 A0 E3 01 10 A0 E3 91 02 01 E0 01 20 42 E2 . . . . B
0x00010010: . . . . 52 E3 00 00 . . 0A FA FF FF EA . . . . R
0x00010020: . . . . 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
0x00010040: . . . . 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
0x00010060: . . . .
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

gcc --version

python3 --version

bash --version

The terminal window shows the following command executions:

- Setting up default-jdk-headless (2:1.21-75+exp1) ...
- Setting up default-jdk (2:1.21-75+exp1) ...
- javac 21.0.9
- java 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)
- 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 3.12.3
- bash 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>>

Assignment 4.3: Compile

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

fib.py / fib.sh

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 / fib.sh

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

C

How do I run a Java program?

Typ in de terminal: javac 'filename'.java

deze command compileert de source code naar bytecode

Typ vervolgens java 'filename'

dit runt de code

How do I run a Python program?

Typ in de terminal: python3 'filename'.py

Python leest de source code en voert die dan uit

How do I run a C program?

Typ in de terminal: gcc -o fib fib.c

Dit compileert de source code en met -o geef je een naam aan het bestand de gecompileerde code

Typ vervolgens ./fib

dit runt de code

How do I run a Bash script?

Typ in de terminal: chmod +x fib.sh

Dit geeft veranderd de permissions van de file

Daarna kun je de source code runnen met ./fib.sh

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

C: Ja, de naam in dit geval fib, de naam bepaal je zelf met -o

Java: Ja, volledige filename + .class

Bash: Nee

Python: Nee

Take relevant screenshots of the following commands:

- Compile the source files where necessary / Make them executable / Run them

Java

```
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.13 milliseconds
```

C

```
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$ gcc -o fib fib.c
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
```

- Which (compiled) source code file performs the calculation the fastest?
C, 0.02 milliseconds

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.

```
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$ gcc --help=optimizer
The following options control optimizations:
  -O<number>           Set optimization level to <number>.
  -Ofast                Optimize for speed disregarding exact standards compliance.
  -Og                   Optimize for debugging experience rather than speed or size.
  -Os                  Optimize for space rather than speed.
  -Oz                  Optimize for space aggressively rather than speed.
  -faggressive-loop-optimizations Aggressively optimize loops using language constraints.
```

- 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?

```
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$ gcc -O2 -o fib fib.c
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds
ruben@ruben-VMware-Virtual-Platform:~/Downloads/code$
```

Ja het is nu sneller

- 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: 0.01 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.22 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.30 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Excution time 5851 milliseconds

ruben@ruben-VMware-Virtual-Platform:~/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 r0, #1  
mov r1, #2  
mov r2, #4
```

Loop:

```
mul r0, r0, r1  
sub r2, r2, #1  
cmp r2, #0  
beq End  
b Loop
```

End:

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

Screenshot of the completed code here.

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