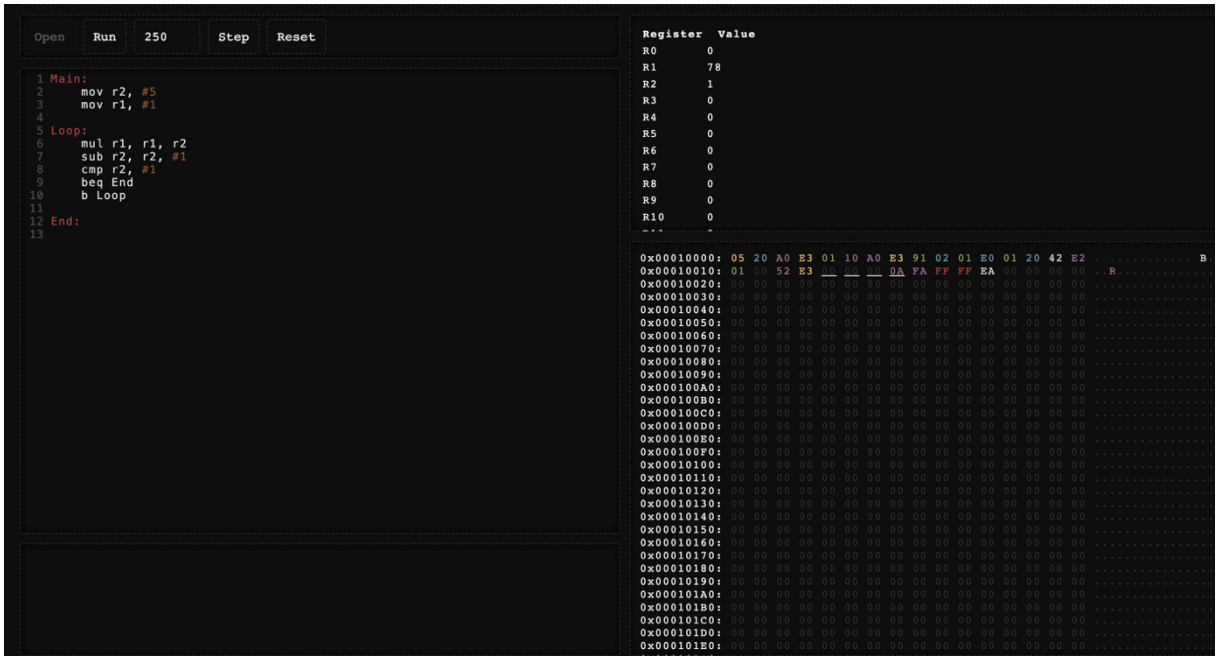
Template Week 4 – Software

Student number:

Assignment 4.1: ARM assembly

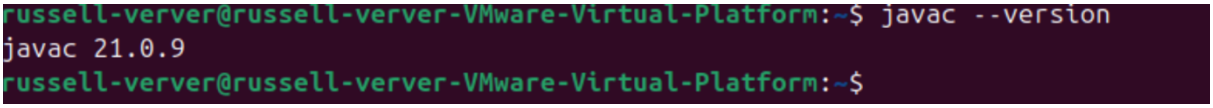
Screenshot of working assembly code of factorial calculation:



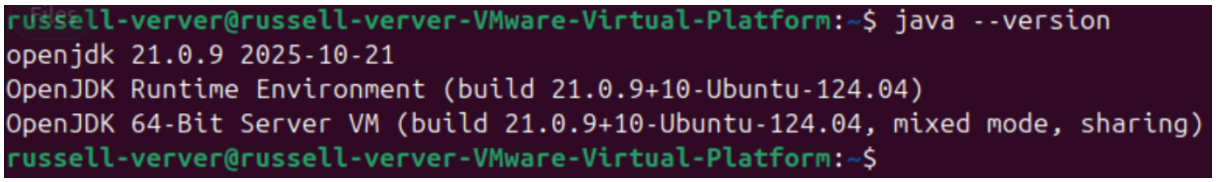
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

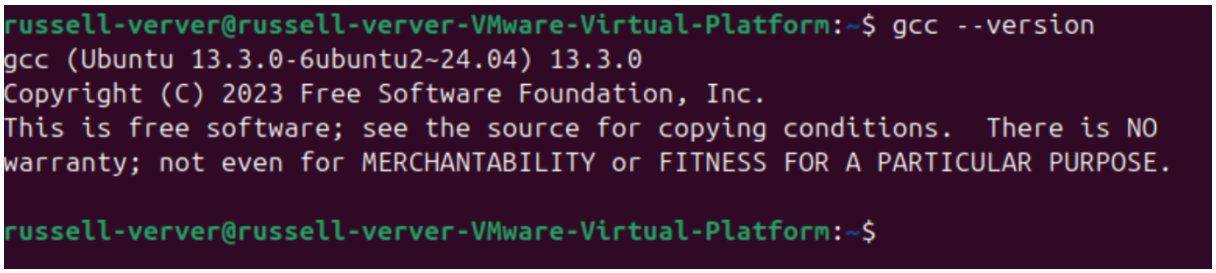
javac --version



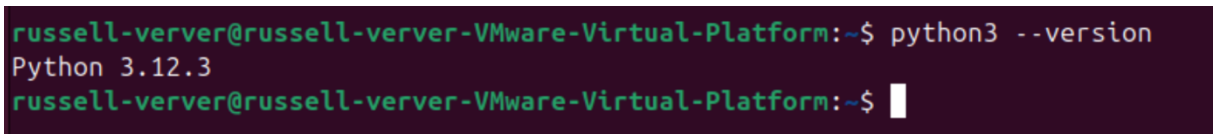
java --version



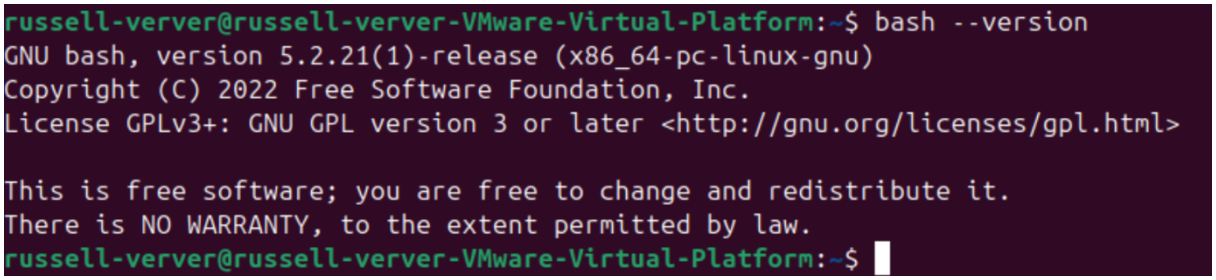
gcc --version



python3 --version



bash --version



Assignment 4.3: Compile

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

. Fib.c

. 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

Which source code files are interpreted by an interpreter?

.Fip.py

.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?

Javac Fibonacci.java

Compileren → How do I run a Python program?

Uitvoeren → java Fibonacci

How do I run a C program?

Compileren → gcc fib.c -o fib

Uitvoeren → ./fib

How do I run a Bash script?

Eerst → chmod +x fib.sh

Uitvoeren → ./fib.sh

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

.C → ja er word een uitvoerbaar bestand aan gemaakt

.java → ja er word een .class bestand aangemaakt

.py → deze code word niet gecompileerd

.sh → deze code word ook niet gecomplieerd

Take relevant screenshots of the following commands:

- Compile the source files where necessary

Java

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$
```

C

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ javac Fibonacci.java
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ls
fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
```

- Make them executable

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.16 milliseconds
```

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

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.22 milliseconds
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$
```

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 4053 milliseconds
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$
```

. Wich (compiled) source code file performs the calculation the fastest?

Dat was de code van c deze deed er maar 0.01 milliseconde over terwijl bash er 4 seconde over deed.

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.

-01

-02

-03

Hoe hoger het het getal aangeeft hoe meer optimalisaties de compiler uitvoert.

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

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ gcc -o3 fib.c -o f
ib_3
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ls
fib  fib_3  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ./fib_3
```

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

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ./fib_3
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$
```

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.

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ time ./fib_3 javac Fibonacci.java time java Fibo
nacci time python3 fib.py time bash fib.sh
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.000s
sys     0m0.002s
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ chmod +x runall.sh
```

```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.02 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.21 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.38 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 6770 milliseconds


russell-verver@russell-verver-VMware-Virtual-Platform:~/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.

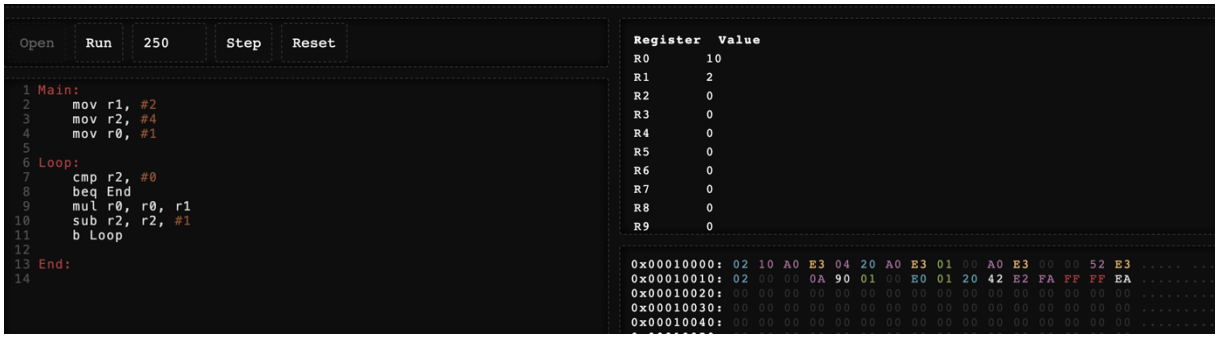
```
Main:
mov r1, #2
mov r2, #4

Loop:

End:
```

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

Screenshot of the completed code here.



Ready? Save this file and export it as a pdf file with the name: **week4.pdf**