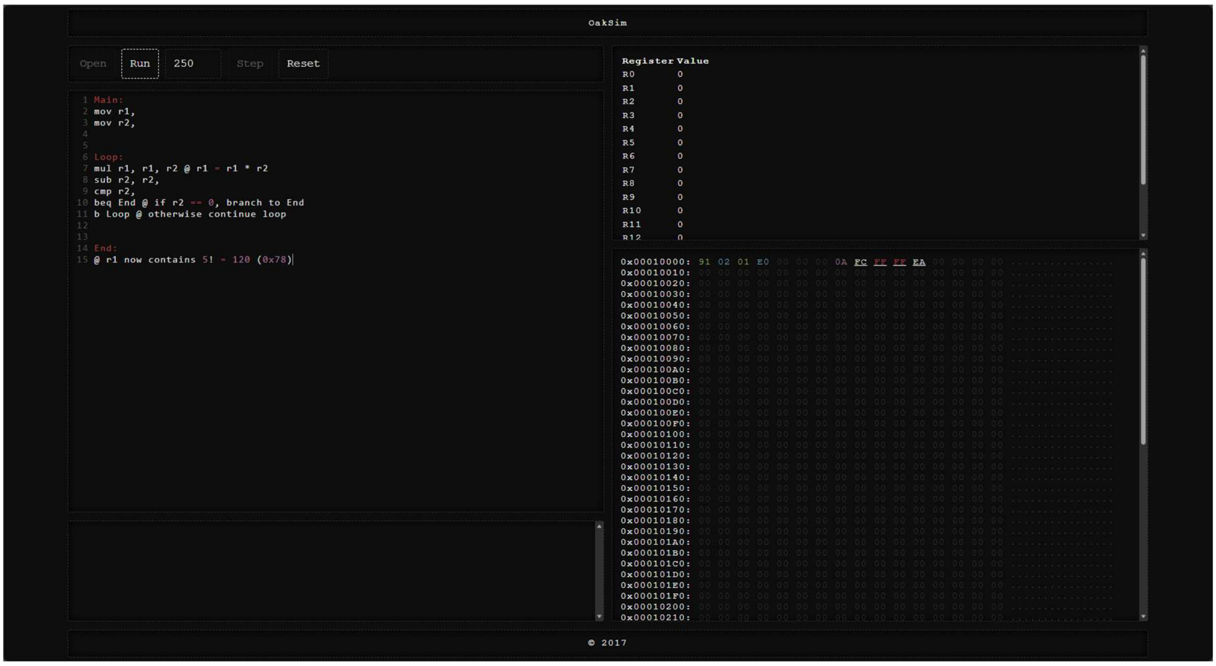
Template Week 4 – Software

Student number: 586377

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?
Fibonacci.java & fib.c

Which source code files are compiled into machine code and then directly executable by a processor?
fib.c needs to be compiled by gcc into a machine-code executable.

Which source code files are compiled to byte code?
Fibonacci.java needs to be compiled by javac into Java bytecode (.class file)

Which source code files are interpreted by an interpreter?
fib.py needs to be interpreted by the Python interpreter (python3)

fib.sh needs to be interpreted by the Bash shell (bash)

These source code files will perform the same calculation after compilation/interpretation. Which one is expected to do the calculation the fastest?
Generally fastest → C program
Because:

- C is compiled into native machine code
- Runs directly on the CPU
- No virtual machine or interpreter overhead

Next fastest → Java
Slower → Python, Bash

How do I run a Java program?
First you need to compile and the je kan run the program commands: javac Fibonacci.java & java Fibonacci

How do I run a Python program?
This runs directly with this command: python3 fib.py

How do I run a C program?
First you need to compile and then you can run the program commands: gcc fib.c -o fib & ./fib

How do I run a Bash script?
This runs directly with this command: ./fib.sh

If I compile the above source code, will a new file be created? If so, which file?
It wil produce Fibonacci.class in java.
In C it will produce: fib
And in python and bash it wont produce new files

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?

```
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.25 milliseconds
```

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

```
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.41 milliseconds
```

```
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ chmod +x fib.sh
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 7161 milliseconds
```

gcc is the fasttest

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.

   -O3 was de snelste want dit geeft de maximale optimalisatie.

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

```
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 -march=native fib.c
-o fib_fast
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
```

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

```
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
luuk@luuk-VMware-Virtual-Platform:~/Downloads/code$ ./fib_fast
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds
```

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: 0.24 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.54 milliseconds


Running BASH Script
S
```

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.

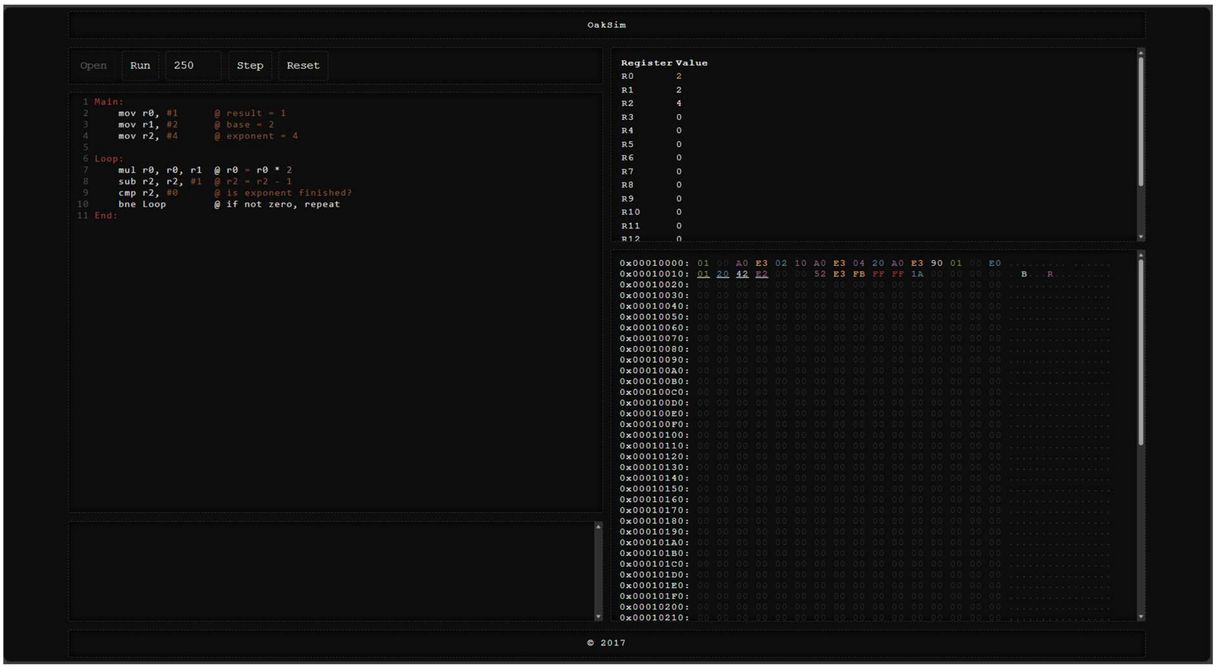
```
Main:
mov r1, #2
mov r2, #4

Loop:
    mul r0, r0, r1  @ r0 = r0 * 2
    sub r2, r2, #1  @ r2 = r2 - 1
    cmp r2, #0      @ is exponent finished?
    bne Loop        @ if not zero, repeat

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**