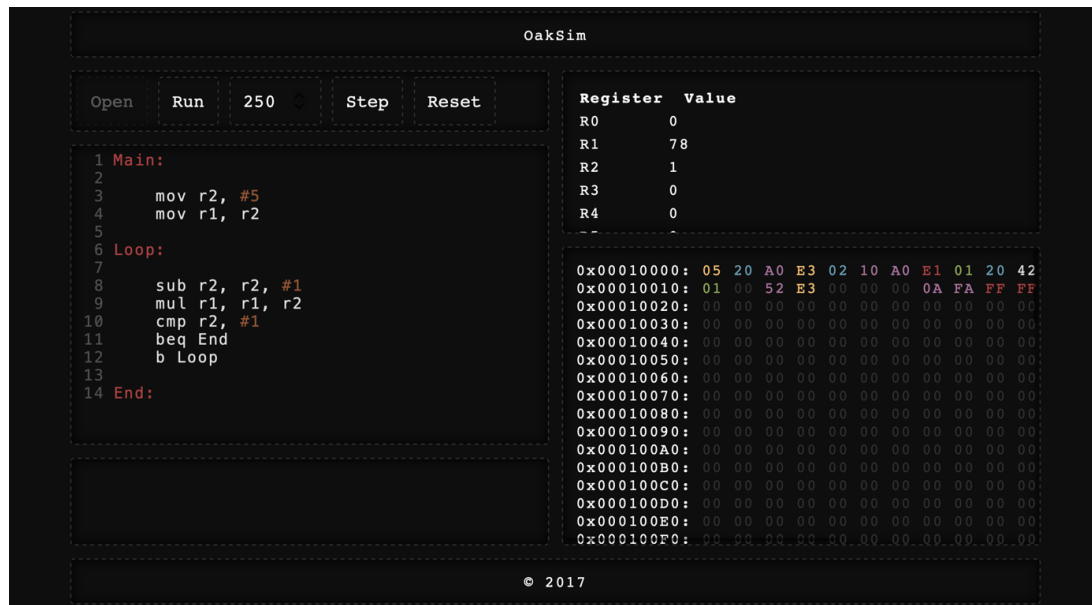


# Template Week 4 – Software

Student number: 585303

## Assignment 4.1: ARM assembly

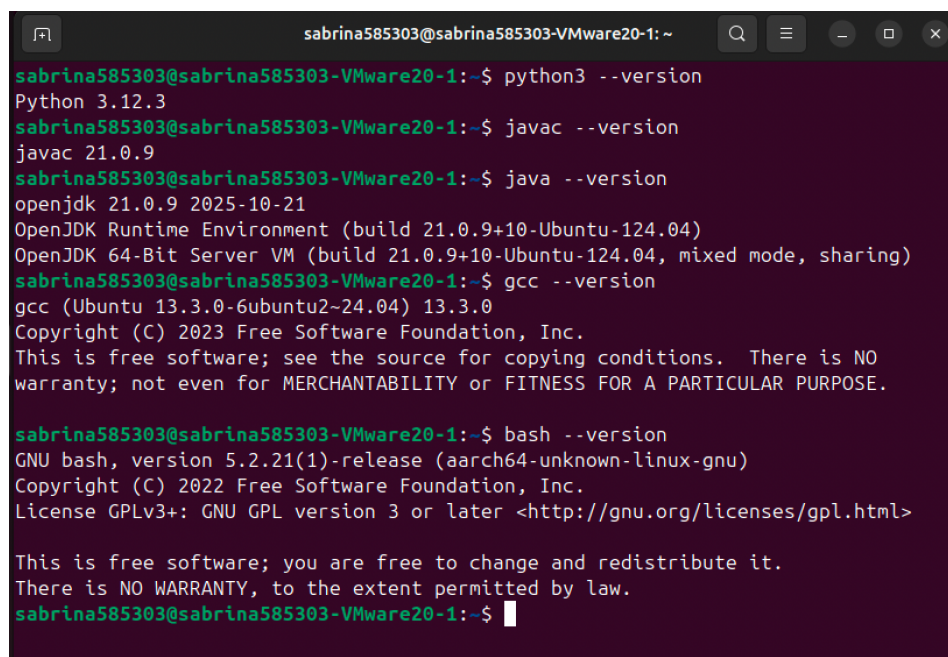
Screenshot of working assembly code of factorial calculation:



## 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

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?

- Fib.c will be the fastest because it is being translated into machine code by a compiler, which makes it able to be executed directly by the processor.

How do I run a Java program?

- `Javac Fibonacci.java`
- `Java Fibonnaci`

How do I run a Python program?

- `Python3 fib.py`

How do I run a C program?

- `gcc fib.c -o fib`
- `./fib`

How do I run a Bash script?

- `chmod +x fib.sh`
- `./fib.sh`

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

- `Fibonacci.class` for `Fibonacci.java`
- An executable file for `fib.c` and we name it `fib`

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?: the `fib.c`

```
sabrina585303@sabrina585303-VMware20-1: ~/Downloads/code
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ chmod +x fib.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 3632 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.64 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/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.
  - I will use the parameter gcc: -O3.
  - It is a strong optimization level that tries to maximize the speed of the compiled program.
  - gcc -O3 fib.c -o fib**
- Compile **fib.c** again with the optimization parameters

```
sabrina585303@sabrina585303-VMware20-1: ~/Downloads/code
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ man gcc
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ gcc -O3 fib.c -o fib
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ time ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds

real    0m0.003s
user    0m0.000s
sys     0m0.003s
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$
```

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

```
sabrina585303@sabrina585303-VMware20-1: ~/Downloads/code
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ man gcc
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ gcc -O3 fib.c -o fib
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ time ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.001s
sys     0m0.001s
sabrina585303@sabrina585303-VMware20-1:~/Downloads/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.

```
sabrina585303@sabrina585303-VMware20-1: ~/Downloads/code
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ chmod +x runall.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ./runall.sh

Running Fibonacci in C
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds

Running Fibonacci Java
Fibonacci(18) = 2584
Execution time: 0.15 milliseconds

Running Fibonacci in Python
Fibonacci(18) = 2584
Execution time: 0.26 milliseconds

Running Fibonacci in Bash
Fibonacci(18) = 2584
Execution time 2178 milliseconds

sabrina585303@sabrina585303-VMware20-1:~/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:

