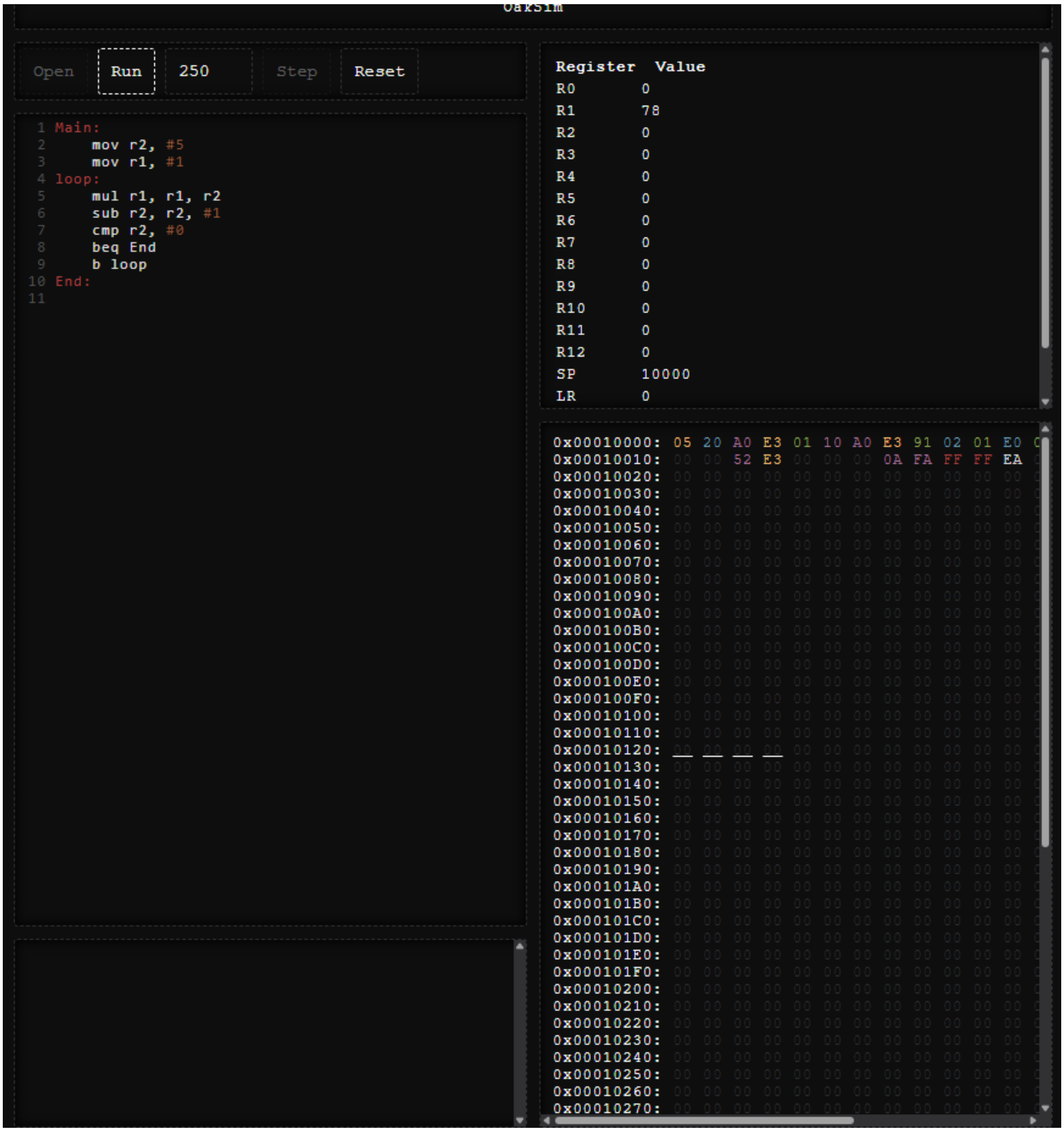
Template Week 4 – Software

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

```
student-581687@student-581687-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
student-581687@student-581687-VMware-Virtual-Platform:~$ java --version
openjdk 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)
student-581687@student-581687-VMware-Virtual-Platform:~$
```

gcc –version

```
student-581687@student-581687-VMware-Virtual-Platform:~$ 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.

student-581687@student-581687-VMware-Virtual-Platform:~$
```

python3 –version

```
student-581687@student-581687-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
student-581687@student-581687-VMware-Virtual-Platform:~$
```

bash –version

```
student-581687@student-581687-VMware-Virtual-Platform:~$ 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.
student-581687@student-581687-VMware-Virtual-Platform:~$
```

Assignment 4.3: Compile

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

Fib.c en 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?

Fib.py door python en fib.sh door bash

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

Fib.c omdat het direct machinecode is.

How do I run a Java program?

Eerst compileren. Dus je maakt van .java een .class en runt het met java …..

How do I run a Python program?

Python3 fib.py

How do I run a C program?

Eerst compileren. Gcc fib.c -o fib. Runt vervolgens ./fib

How do I run a Bash script?

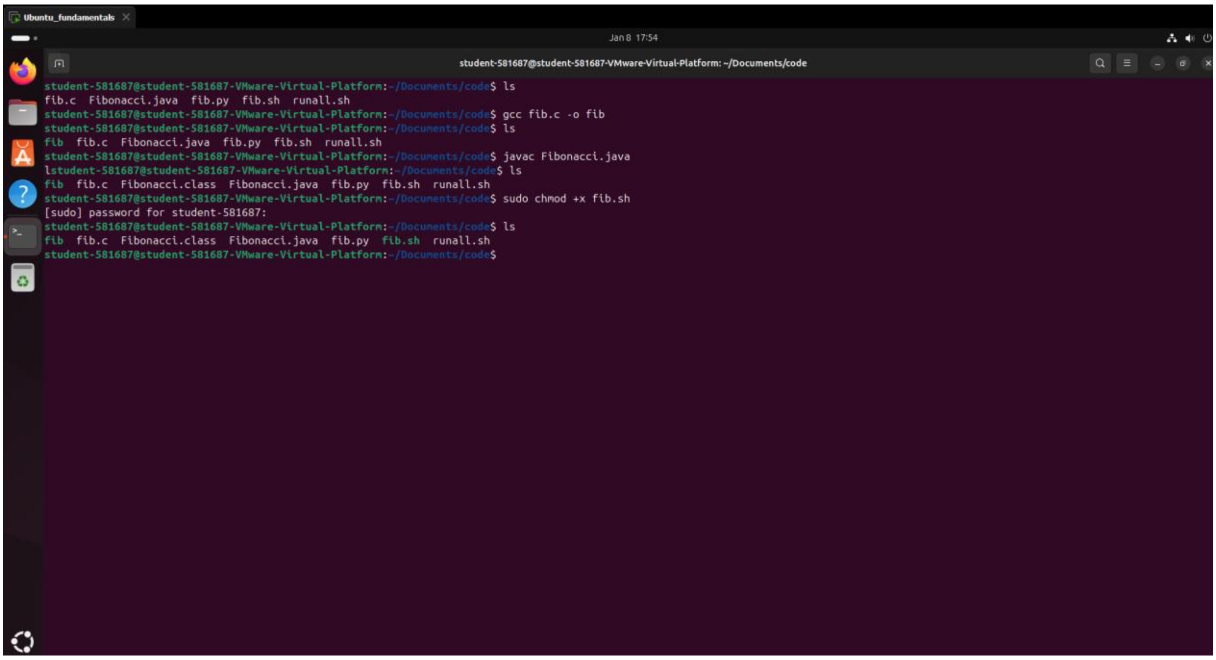
Sudo chmod a+x fib.sh en dan sudo ./fib.sh

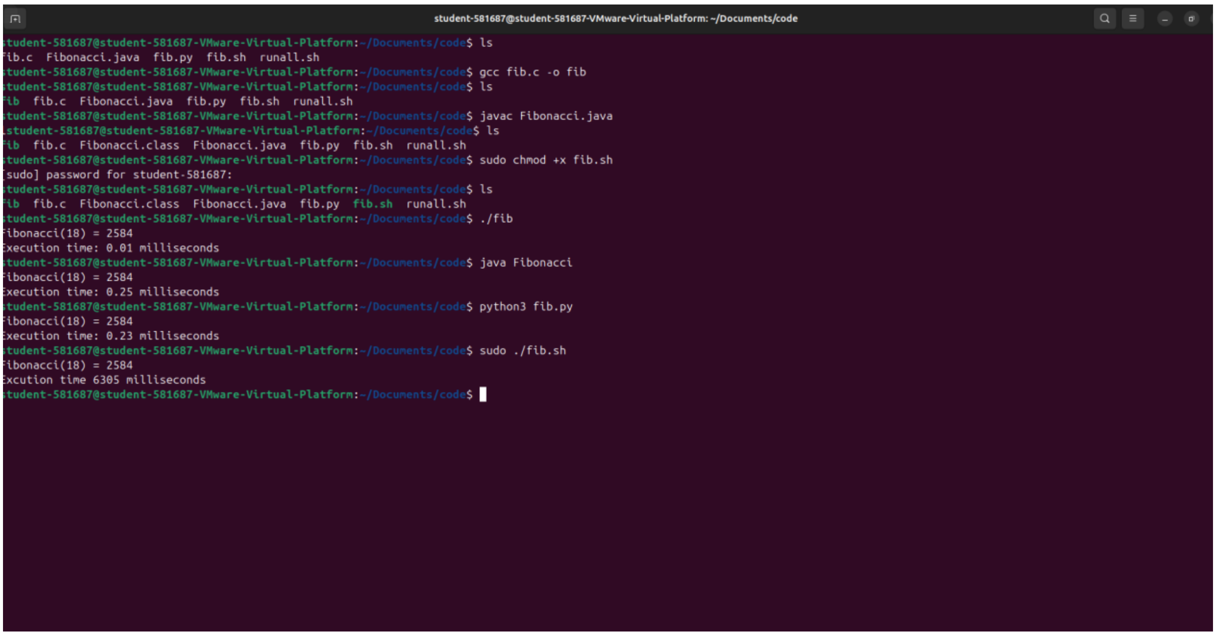
If I compile the above source code, will a new file be created? If so, which file?

Bij C(fib) en java(class)

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?





Fib (C) is de snelste met 0.01ms

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.

```
Optimization Options
    -faggressive-loop-optimizations -falign-functions[=n[:m:[n2[:m2]]]]  -falign-jumps[=n[:m:[n2[:m2]]]]  -falign-labels[=n[:m:[n2[:m2]]]]  -falign-loops[=n[:m:[n2[:m2]]]]
    -fno-allocation-dce   -fallow-store-data-races   -fassociative-math    -fauto-profile    -fauto-profile[=path]  -fauto-inc-dec   -fbranch-probabilities  -fcaller-saves
    -fcombine-stack-adjustments    -fconserve-stack   -fcompare-elim    -fcprop-registers    -fcrossjumping   -fcse-follow-jumps    -fcse-skip-blocks    -fcx-fortran-rules
    -fcx-limited-range  -fdata-sections  -fdce  -fdelayed-branch -fdelete-null-pointer-checks  -fdevirtualize  -fdevirtualize-speculatively -fdevirtualize-at-ltrans  -fdse
    -fearly-inlining  -fipa-sra  -fexpensive-optimizations   -ffat-lto-objects  -ffast-math   -ffinite-math-only   -ffloat-store   -fexcess-precision=style  -ffinite-loops
    -fforward-propagate  -ffp-contract=style  -ffunction-sections -fgcse  -fgcse-after-reload  -fgcse-las  -fgcse-lm  -fgraphite-identity -fgcse-sm  -fhoist-adjacent-loads
    -fif-conversion  -fif-conversion2   -findirect-inlining  -finline-functions   -finline-functions-called-once   -finline-limit=n  -finline-small-functions  -fipa-modref
    -fipa-cp  -fipa-cp-clone  -fipa-bit-cp   -fipa-vrp   -fipa-pta   -fipa-profile   -fipa-pure-const  -fipa-reference   -fipa-reference-addressable  -fipa-stack-alignment
    -fipa-icf      -fira-algorithm=algorithm      -flive-patching=level     -fira-region=region      -fira-hoist-pressure     -fira-loop-pressure      -fno-ira-share-save-slots
    -fno-ira-share-spill-slots  -fisolate-erroneous-paths-dereference   -fisolate-erroneous-paths-attribute  -fivopts    -fkeep-inline-functions    -fkeep-static-functions
    -fkeep-static-consts      -flimit-function-alignment      -flive-range-shrinkage      -floop-block      -floop-interchange      -floop-strip-mine     -floop-unroll-and-jam
    -floop-nest-optimize -floop-parallelize-all  -flra-remat  -flto  -flto-compression-level -flto-partition=alg  -fmerge-all-constants  -fmerge-constants   -fmodulo-sched
    -fmodulo-sched-allow-regmoves   -fmove-loop-invariants    -fmove-loop-stores     -fno-branch-count-reg  -fno-defer-pop   -fno-fp-int-builtin-inexact   -fno-function-cse
    -fno-guess-branch-probability   -fno-inline    -fno-math-errno    -fno-peephole   -fno-peephole2    -fno-printf-return-value    -fno-sched-interblock   -fno-sched-spec
    -fno-signed-zeros   -fno-toplevel-reorder    -fno-trapping-math    -fno-zero-initialized-in-bss   -fomit-frame-pointer    -foptimize-sibling-calls   -fpartial-inlining
    -fpeel-loops   -fpredictive-commoning  -fprefetch-loop-arrays  -fprofile-correction  -fprofile-use   -fprofile-use=path   -fprofile-partial-training   -fprofile-values
    -fprofile-reorder-functions   -freciprocal-math    -free    -frename-registers   -freorder-blocks   -freorder-blocks-algorithm=algorithm  -freorder-blocks-and-partition
    -freorder-functions -frerun-cse-after-loop  -freschedule-modulo-scheduled-loops -frounding-math  -fsave-optimization-record -fsched2-use-superblocks   -fsched-pressure
    -fsched-spec-load    -fsched-spec-load-dangerous   -fsched-stalled-insns-dep[=n]   -fsched-stalled-insns[=n]  -fsched-group-heuristic   -fsched-critical-path-heuristic
    -fsched-spec-insn-heuristic  -fsched-rank-heuristic  -fsched-last-insn-heuristic   -fsched-dep-count-heuristic  -fschedule-fusion  -fschedule-insns   -fschedule-insns2
    -fsection-anchors  -fselective-scheduling   -fselective-scheduling2  -fsel-sched-pipelining  -fsel-sched-pipelining-outer-loops -fsemantic-interposition  -fshrink-wrap
    -fshrink-wrap-separate -fsignaling-nans -fsingle-precision-constant  -fsplit-ivs-in-unroller  -fsplit-loops -fsplit-paths -fsplit-wide-types   -fsplit-wide-types-early
    -fssa-backprop   -fssa-phiopt  -fstdarg-opt   -fstore-merging  -fstrict-aliasing -fipa-strict-aliasing -fthread-jumps  -ftracer  -ftree-bit-ccp -ftree-builtin-call-dce
    -ftree-ccp   -ftree-ch  -ftree-coalesce-vars   -ftree-copy-prop    -ftree-dce    -ftree-dominator-opts   -ftree-dse    -ftree-forwprop    -ftree-fre    -fcode-hoisting
    -ftree-loop-if-convert     -ftree-loop-im    -ftree-phiprop     -ftree-loop-distribution    -ftree-loop-distribute-patterns   -ftree-loop-ivcanon    -ftree-loop-linear
    -ftree-loop-optimize -ftree-loop-vectorize  -ftree-parallelize-loops=n   -ftree-pre   -ftree-partial-pre   -ftree-pta  -ftree-reassoc   -ftree-scev-cprop   -ftree-sink
    -ftree-slsr    -ftree-sra   -ftree-switch-conversion    -ftree-tail-merge  -ftree-ter   -ftree-vectorize   -ftree-vrp   -ftrivial-auto-var-init  -funconstrained-commons
    -funit-at-a-time  -funroll-all-loops -funroll-loops -funsafe-math-optimizations  -funswitch-loops -fipa-ra  -fvariable-expansion-in-unroller  -fvect-cost-model   -fvpt
    -fweb  -fwhole-program  -fwpa  -fuse-linker-plugin -fzero-call-used-regs --param name=value -O  -O0  -O1  -O2  -O3  -Os  -Ofast  -Og  -Oz
```

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

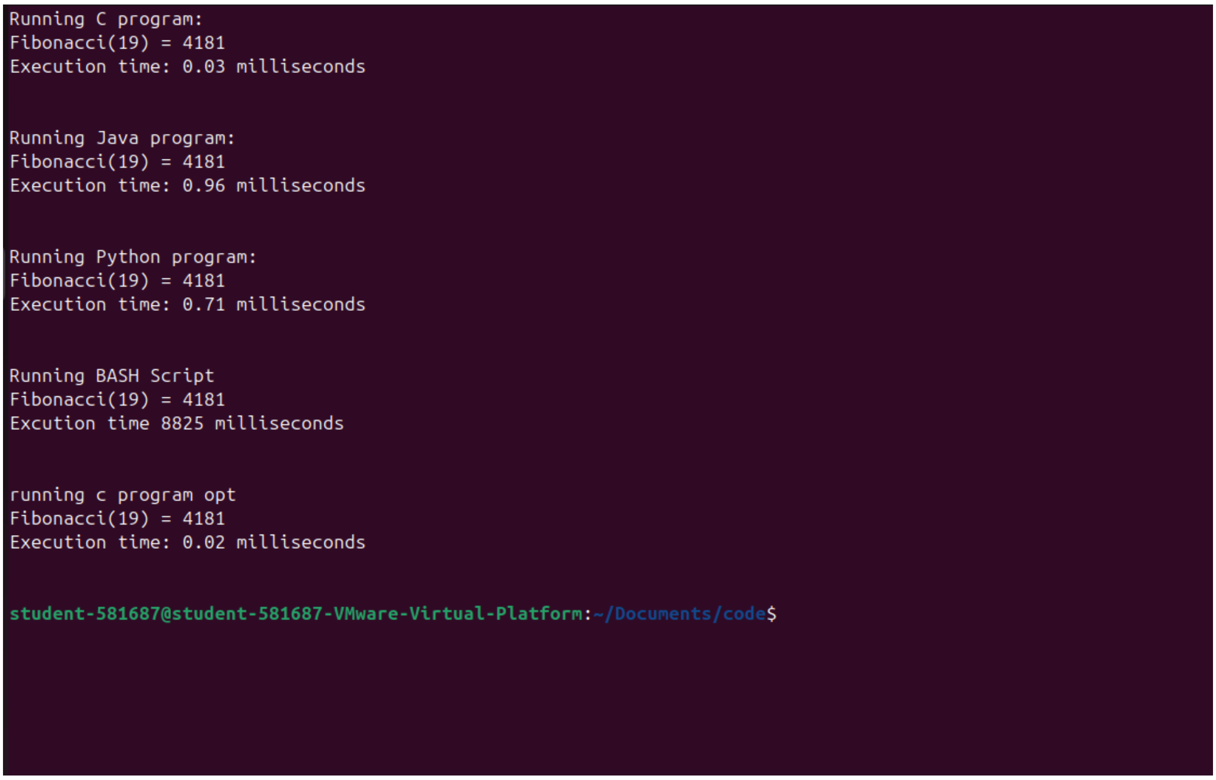
```
student-581687@student-581687-VMware-Virtual-Platform:~/Documents/code$ gcc -o3 fib.c -o fib_opt
student-581687@student-581687-VMware-Virtual-Platform:~/Documents/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib_opt  fib.py  fib.sh  runall.sh
student-581687@student-581687-VMware-Virtual-Platform:~/Documents/code$ ls -l
```

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

No it isnt.

```
student-581687@student-581687-VMware-Virtual-Platform:~/Documents/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
student-581687@student-581687-VMware-Virtual-Platform:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.08 milliseconds
student-581687@student-581687-VMware-Virtual-Platform:~/Documents/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.

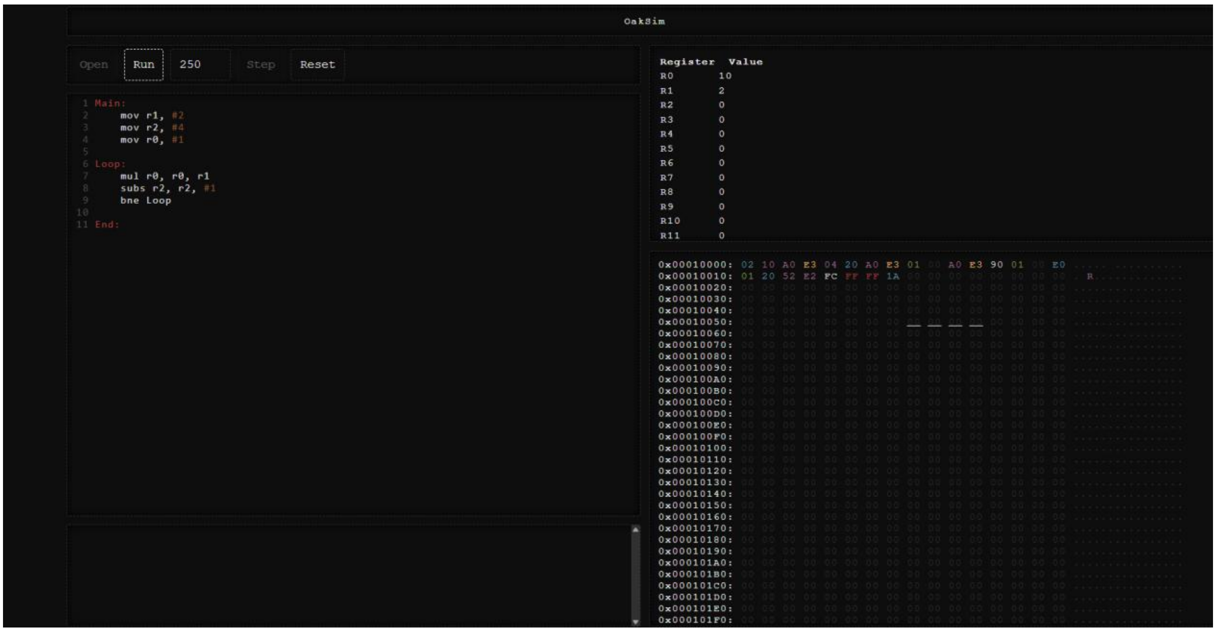


Assignment 4.5: More ARM Assembly

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.



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