Template Week 4 – Software

Student number:

582762

Assignment 4.1: ARM assembly

Enter the following code and run it by clicking the RUN button. You can also stop the program by clicking the RUN button again.

```
Main:
    add r0, r0, #2
    mul r1, r0, r0
    sub r2, r1, r0
    mov r3, #15
```

What is the value of:

- r0?
- r1?
- r2?
- r3?

Na één keer dit programma runnen krijg ik

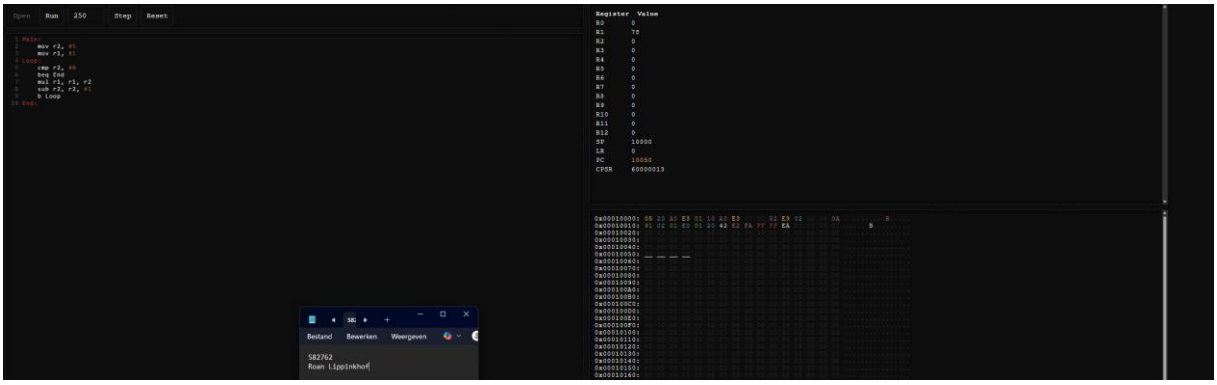
R0 = 1

R1 = 1

R2 = 0

R3 = 0

Screenshot of working assembly code of factorial calculation:



Assignment 4.2: Programming languages

Take screenshots that the following commands work:

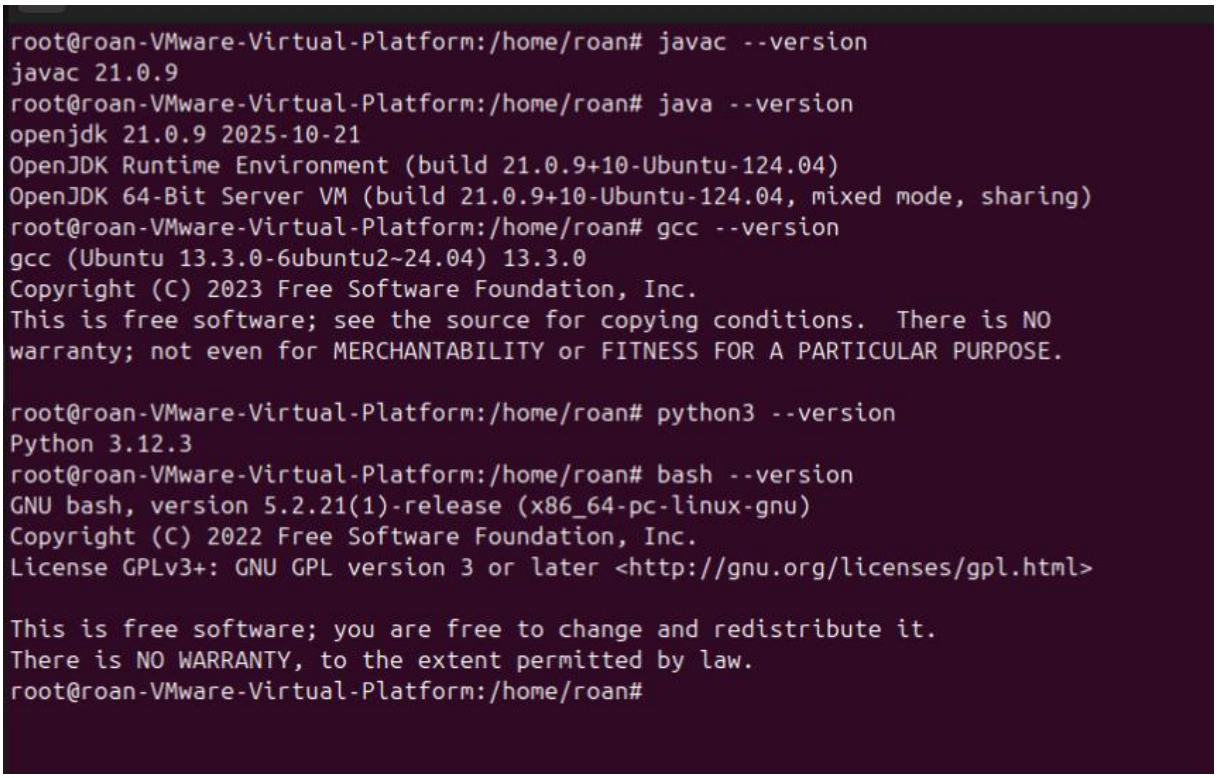
javac --version

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?

Fibonaci.java en 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 omdat deze direct uitvoerbaar is door de processor nadat hij is gecompileerd.

How do I run a Java program?

Javac Fibonacci.java (compileren) -> java Fibonacci (runnen)

How do I run a Python program?

Python3 fib.py

How do I run a C program?

Gcc fib.c -o fib (compileren) -> ./fib (runnen)

How do I run a Bash script?

chmod a+x fib.sh (uitvoerbaar maken) -> ./fib.sh (runnen)

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

Fibonacci.java -> Fibonacci.class

Fib.c -> fib

Fib.py -> geen nieuwe naam

Fib.sh -> geen nieuwe naam

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?

```
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ls -l
total 16
-rw-rw-r-- 1 roan roan 831 Jun  9  2023 fib.c
-rw-rw-r-- 1 roan roan 839 Jun  9  2023 Fibonacci.java
-rw-rw-r-- 1 roan roan 516 Jun  9  2023 fib.py
-rw-rw-r-- 1 roan roan 249 Jun  9  2023 runall.sh
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# S
```

Hierboven zie je de huidige map met de 4 bestanden

```
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# javac Fibonacci.java
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.25 milliseconds
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ls -l
total 20
-rw-rw-r-- 1 roan roan  831 Jun  9  2023 fib.c
-rw-r--r-- 1 root root 1448 Jan  2 16:34 Fibonacci.class
-rw-rw-r-- 1 roan roan  839 Jun  9  2023 Fibonacci.java
-rw-rw-r-- 1 roan roan  516 Jun  9  2023 fib.py
-rw-rw-r-- 1 roan roan  249 Jun  9  2023 runall.sh
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code#
```

Hierboven zie je dat ik java heb gebruikt met ook de execution time en het nieuwe bestands naam

```
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.39 milliseconds
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ls -l
total 16
-rw-rw-r-- 1 roan roan 831 Jun  9  2023 fib.c
-rw-rw-r-- 1 roan roan 839 Jun  9  2023 Fibonacci.java
-rw-rw-r-- 1 roan roan 516 Jun  9  2023 fib.py
-rw-rw-r-- 1 roan roan 249 Jun  9  2023 runall.sh
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code#
```

Hierboven zie je dat ik python heb gebruikt met de daarbijhorende execution time

```
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# gcc fib.c -o fib
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ls -l
total 32
-rwxr-xr-x 1 root root 16136 Jan  2 16:37 fib
-rw-rw-r-- 1 roan roan   831 Jun  9  2023 fib.c
-rw-rw-r-- 1 roan roan   839 Jun  9  2023 Fibonacci.java
-rw-rw-r-- 1 roan roan   516 Jun  9  2023 fib.py
-rw-rw-r-- 1 roan roan   249 Jun  9  2023 runall.sh
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# S
```

Hierboven zie je dat ik C heb gebruikt om te compileren en runnen met de daarbijbehorende execution time.

```
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# chmod a+x fib.sh
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ./fib.sh
Fibonacci(18) = 2584
Excution time 6083 milliseconds
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ls -l
total 20
-rw-rw-r-- 1 roan roan 831 Jun  9  2023 fib.c
-rw-rw-r-- 1 roan roan 839 Jun  9  2023 Fibonacci.java
-rw-rw-r-- 1 roan roan 516 Jun  9  2023 fib.py
-rwxrwxr-x 1 roan roan 668 Jun  9  2023 fib.sh
-rw-rw-r-- 1 roan roan 249 Jun  9  2023 runall.sh
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code#
```

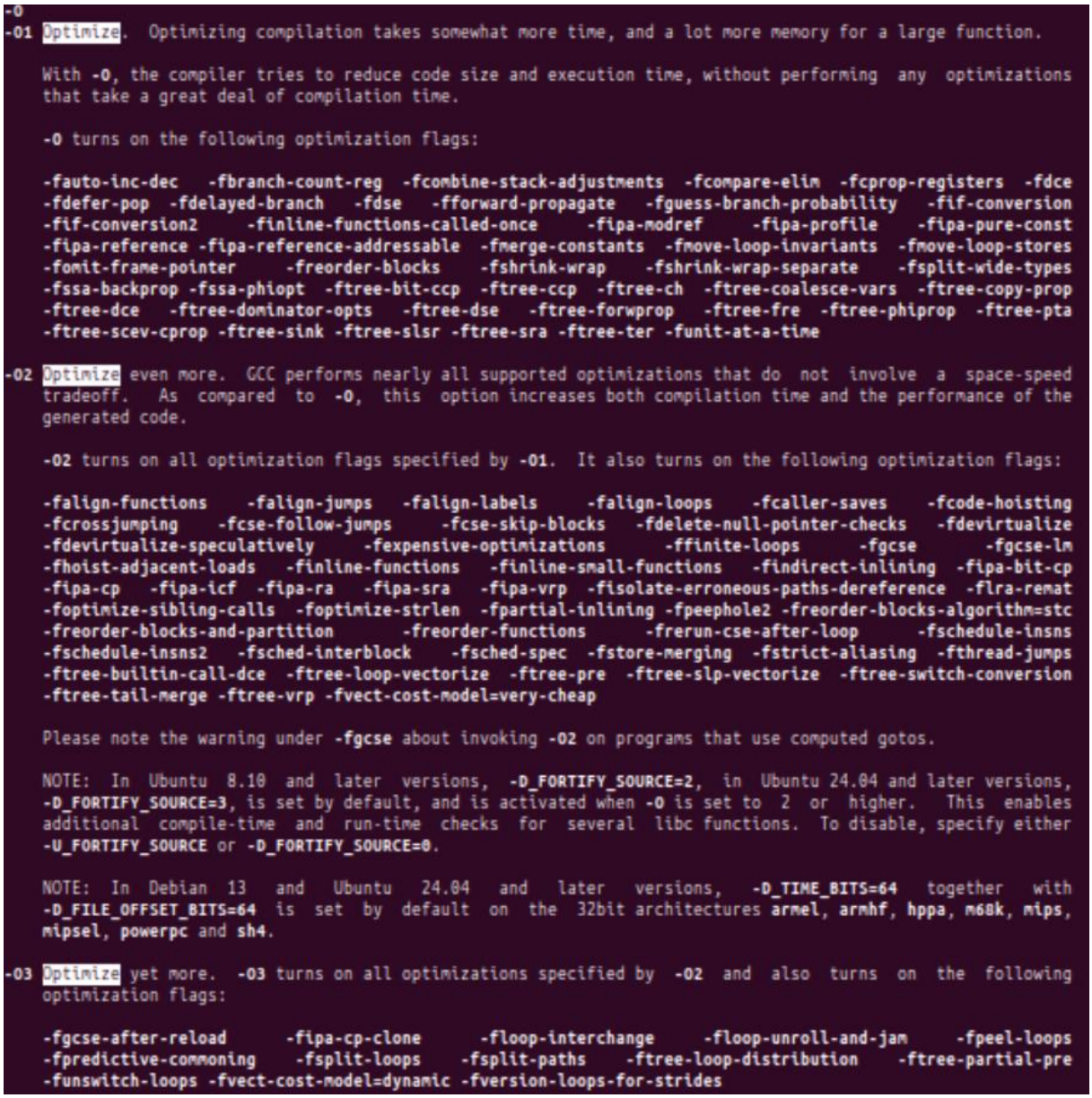
Hierboven zie je dat ik bash heb gebruikt met de bijbehorende execution time

Op alle vier van de afbeeldingen kun je zien welke het langzaamst is en welke het snelst is.

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.



```
-O
-O1 Optimize.  Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

    With -O, the compiler tries to reduce code size and execution time, without performing  any  optimizations
    that take a great deal of compilation time.

    -O turns on the following optimization flags:

    -fauto-inc-dec  -fbranch-count-reg  -fcombine-stack-adjustments  -fcompare-elim  -fcprop-registers  -fdce
    -fdefer-pop  -fdelayed-branch  -fdse  -fforward-propagate  -fguess-branch-probability  -fif-conversion
    -fif-conversion2  -finline-functions-called-once  -fipa-modref  -fipa-profile  -fipa-pure-const
    -fipa-reference -fipa-reference-addressable  -fmerge-constants  -fmove-loop-invariants  -fmove-loop-stores
    -fomit-frame-pointer  -freorder-blocks  -fshrink-wrap  -fshrink-wrap-separate  -fsplit-wide-types
    -fssa-backprop -fssa-phiopt  -ftree-bit-ccp  -ftree-ccp  -ftree-ch  -ftree-coalesce-vars  -ftree-copy-prop
    -ftree-dce  -ftree-dominator-opts  -ftree-dse  -ftree-forwprop  -ftree-fre  -ftree-phiprop  -ftree-pta
    -ftree-scev-cprop -ftree-sink -ftree-slsr -ftree-sra -ftree-ter -funit-at-a-time

-O2 Optimize even more.  GCC performs nearly all supported optimizations that do  not  involve  a  space-speed
    tradeoff.  As  compared  to  -O,  this  option increases both compilation time and the performance of the
    generated code.

    -O2 turns on all optimization flags specified by -O1.  It also turns on the following optimization flags:

    -falign-functions  -falign-jumps  -falign-labels  -falign-loops  -fcaller-saves  -fcode-hoisting
    -fcrossjumping  -fcse-follow-jumps  -fcse-skip-blocks  -fdelete-null-pointer-checks  -fdevirtualize
    -fdevirtualize-speculatively  -fexpensive-optimizations  -ffinite-loops  -fgcse  -fgcse-lm
    -fhoist-adjacent-loads  -finline-functions  -finline-small-functions  -findirect-inlining  -fipa-bit-cp
    -fipa-cp  -fipa-icf  -fipa-ra  -fipa-sra  -fipa-vrp  -fisolate-erroneous-paths-dereference  -flra-remat
    -foptimize-sibling-calls  -foptimize-strlen  -fpartial-inlining -fpeephole2 -freorder-blocks-algorithm=stc
    -freorder-blocks-and-partition  -freorder-functions  -frerun-cse-after-loop  -fschedule-insns
    -fschedule-insns2  -fsched-interblock  -fsched-spec  -fstore-merging  -fstrict-aliasing  -fthread-jumps
    -ftree-builtin-call-dce  -ftree-loop-vectorize  -ftree-pre  -ftree-slp-vectorize  -ftree-switch-conversion
    -ftree-tail-merge -ftree-vrp -fvect-cost-model=very-cheap

    Please note the warning under -fgcse about invoking -O2 on programs that use computed gotos.

    NOTE:  In  Ubuntu  8.10  and  later  versions,  -D_FORTIFY_SOURCE=2,  in  Ubuntu 24.04 and later versions,
    -D_FORTIFY_SOURCE=3, is set by default, and is activated when -O is set to  2  or  higher.  This  enables
    additional  compile-time  and  run-time  checks  for  several  libc functions.  To disable, specify either
    -U_FORTIFY_SOURCE or -D_FORTIFY_SOURCE=0.

    NOTE:  In  Debian  13  and  Ubuntu  24.04  and  later  versions,  -D_TIME_BITS=64  together  with
    -D_FILE_OFFSET_BITS=64  is  set  by  default  on  the  32bit architectures armel, armhf, hppa, m68k, mips,
    mipsel, powerpc and sh4.

-O3 Optimize yet more.  -O3 turns on all optimizations specified by  -O2  and  also  turns  on  the  following
    optimization flags:

    -fgcse-after-reload  -fipa-cp-clone  -floop-interchange  -floop-unroll-and-jam  -fpeel-loops
    -fpredictive-commoning  -fsplit-loops  -fsplit-paths  -ftree-loop-distribution  -ftree-partial-pre
    -funswitch-loops -fvect-cost-model=dynamic -fversion-loops-for-strides
```

Hierboven zie je dat ik info gcc heb gebruikt om de parameters te bekijken welke het snelst is. Ik kon het niet zo snel vinden in man of met gebruik van –help dus heb ik info maar gebruikt. In dit geval is -O3 het snelst.

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

```
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# gcc fib.c -o fib
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# gcc -O3 fib.c -o fib_optimized
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code# ./fib_optimized
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
root@roan-VMware-Virtual-Platform:/home/roan/Downloads/code#
```

Hier zie je het Verschil in snelheid met de optimization

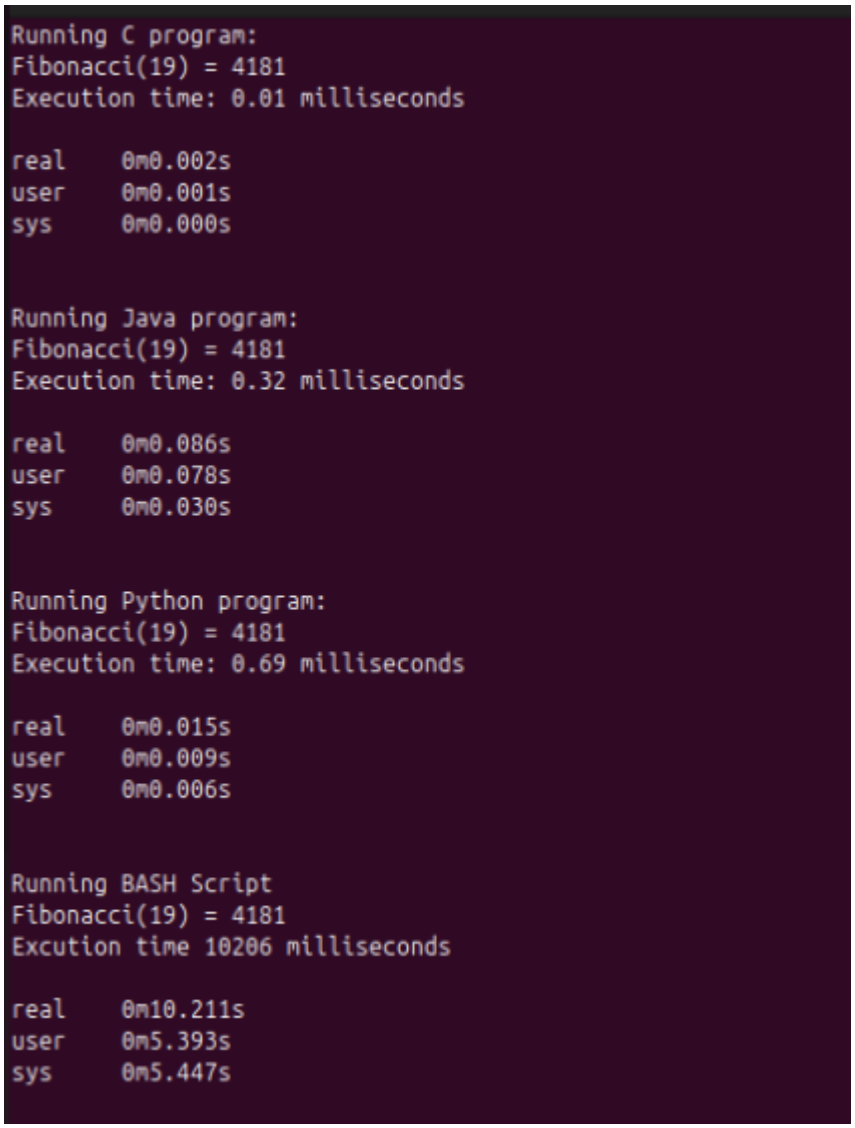
c) Run the newly compiled program. Is it true that it now performs the calculation faster?

Ja, het is nu 0.01 milliseconden 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.

```
 GNU nano 7.2
#!/bin/bash
clear
n=19

echo "Running C program:"
gcc -O3 fib.c -o fib_optimized
time ./fib_optimized $n
echo -e '\n'

echo "Running Java program:"
javac Fibonacci.java
time java Fibonacci $n
echo -e '\n'

echo "Running Python program:"
time python3 fib.py $n
echo -e '\n'

echo "Running BASH Script"
chmod a+x fib.sh
time ./fib.sh $n
echo -e '\n'
```

Hierboven zie je het runall bestand in nano , met alle compilers eerst en daarna worden de Bestanden gerund.



hier zie je de uitkomst van het bestand runall

ik moest het bestand runall nog wel eerst permissie geven met chmod a+x runall.sh

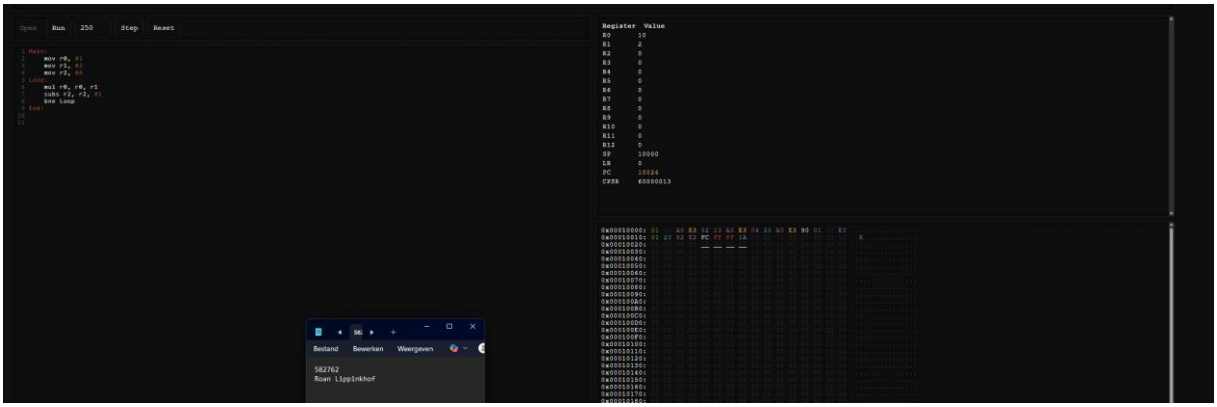
Assignment 4.5: More ARM Assembly

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
    subs r2, r2, #1
    bne 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**