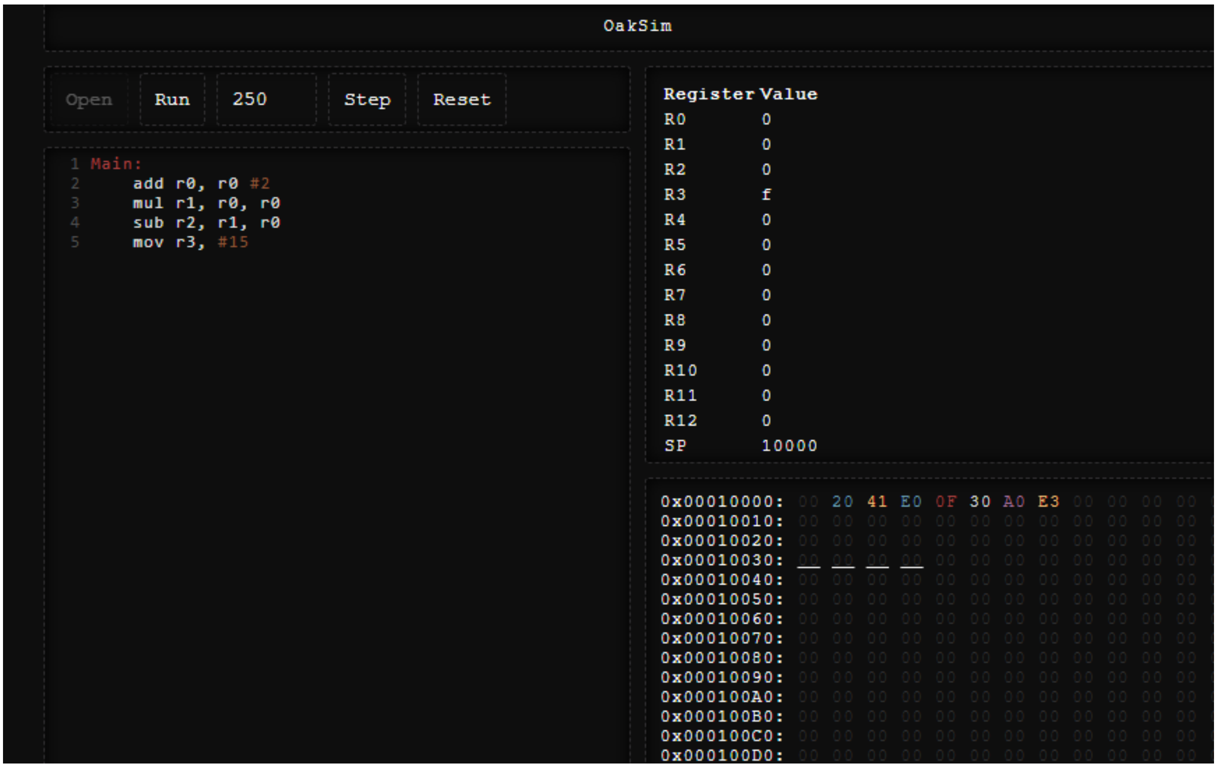
Template Week 4 – Software

Student number:

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



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

```
Register Value
R0       0
R1       0
R2       0
R3       f
R4       0
R5       0
R6       0
R7       0
R8       0
R9       0
R10      0
R11      0
R12      0
SP       10000
```

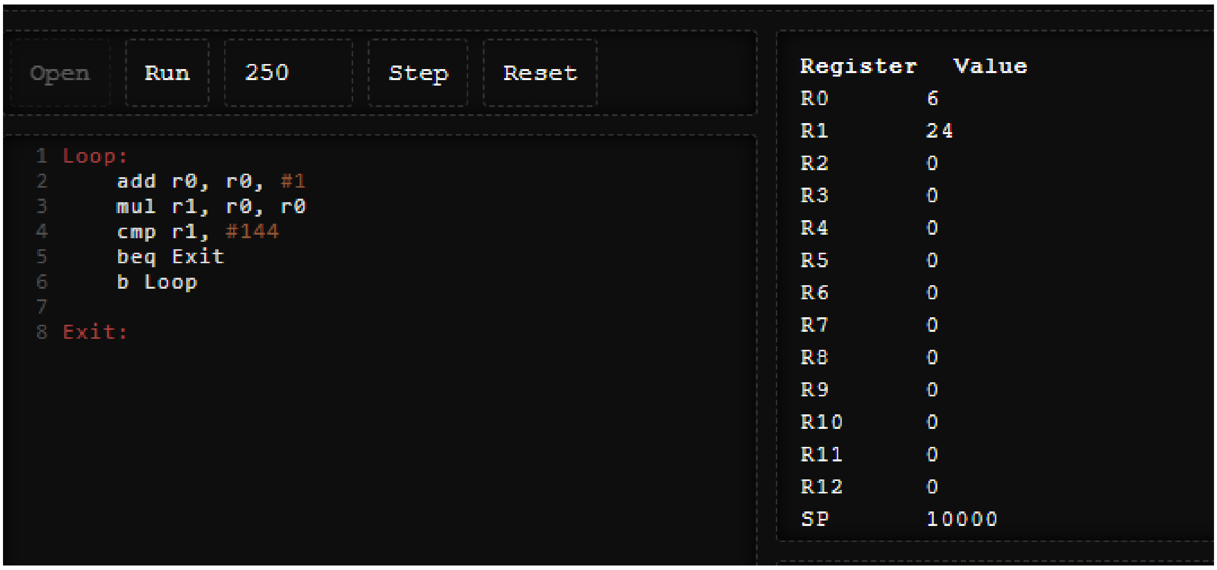
```
0x00010000: 00 20 41 E0 0F 30 A0 E3 00 00 00 00
0x00010010: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010020: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010030: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010040: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010050: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010060: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010070: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010080: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010090: 00 00 00 00 00 00 00 00 00 00 00 00
0x000100A0: 00 00 00 00 00 00 00 00 00 00 00 00
0x000100B0: 00 00 00 00 00 00 00 00 00 00 00 00
0x000100C0: 00 00 00 00 00 00 00 00 00 00 00 00
0x000100D0: 00 00 00 00 00 00 00 00 00 00 00 00
```

De inhoud van de registers worden hexadecimal weergegeven.

R1 = 0

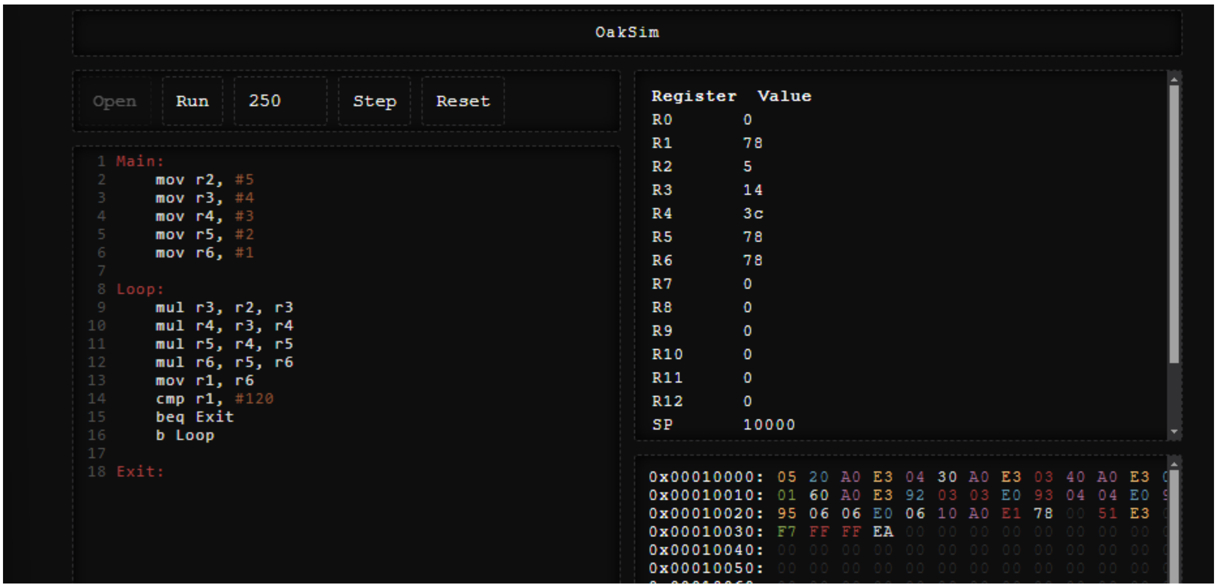
R2 = 0

R3 = F



```
Loop:
    add r0, r0, #1
    mul r1, r0, r0
    cmp r1, #144
    beq Exit
    b Loop

Exit:
```

```
Register   Value
R0         6
R1         24
R2         0
R3         0
R4         0
R5         0
R6         0
R7         0
R8         0
R9         0
R10        0
R11        0
R12        0
SP         10000
```



```
Main:
    mov r2, #5
    mov r3, #4
    mov r4, #3
    mov r5, #2
    mov r6, #1

Loop:
    mul r3, r2, r3
    mul r4, r3, r4
    mul r5, r4, r5
    mul r6, r5, r6
    mov r1, r6
    cmp r1, #120
    beq Exit
    b Loop

Exit:
```

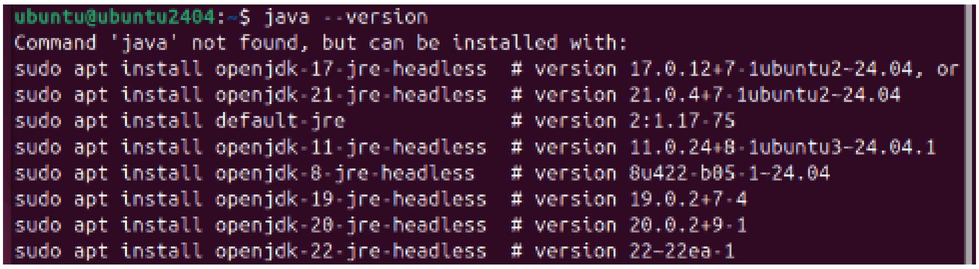
```
Register  Value
R0        0
R1        78
R2        5
R3        14
R4        3c
R5        78
R6        78
R7        0
R8        0
R9        0
R10       0
R11       0
R12       0
SP        10000
```

```
0x00010000: 05 20 A0 E3 04 30 A0 E3 03 40 A0 E3
0x00010010: 01 60 A0 E3 92 03 03 E0 93 04 04 E0
0x00010020: 95 06 06 E0 06 10 A0 E1 78 00 51 E3
0x00010030: F7 FF FF EA 00 00 00 00 00 00 00 00
0x00010040: 00 00 00 00 00 00 00 00 00 00 00 00
0x00010050: 00 00 00 00 00 00 00 00 00 00 00 00
```

Assignment 4.2: Programming languages

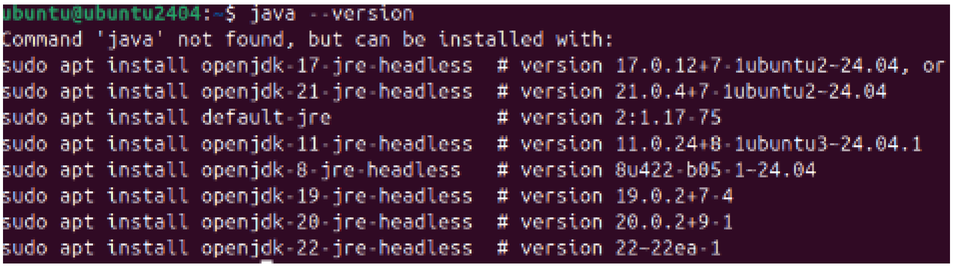
Take screenshots that the following commands work:

javac –version



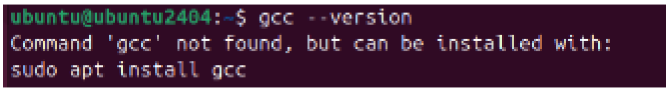
```
ubuntu@ubuntu2404:~$ java --version
Command 'java' not found, but can be installed with:
sudo apt install openjdk-17-jre-headless  # version 17.0.12+7-1ubuntu2-24.04, or
sudo apt install openjdk-21-jre-headless  # version 21.0.4+7-1ubuntu2-24.04
sudo apt install default-jre              # version 2:1.17-75
sudo apt install openjdk-11-jre-headless  # version 11.0.24+8-1ubuntu3-24.04.1
sudo apt install openjdk-8-jre-headless   # version 8u422-b05-1-24.04
sudo apt install openjdk-19-jre-headless  # version 19.0.2+7-4
sudo apt install openjdk-20-jre-headless  # version 20.0.2+9-1
sudo apt install openjdk-22-jre-headless  # version 22-22ea-1
```

java --version



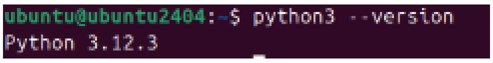
```
ubuntu@ubuntu2404:~$ java --version
Command 'java' not found, but can be installed with:
sudo apt install openjdk-17-jre-headless  # version 17.0.12+7-1ubuntu2-24.04, or
sudo apt install openjdk-21-jre-headless  # version 21.0.4+7-1ubuntu2-24.04
sudo apt install default-jre              # version 2:1.17-75
sudo apt install openjdk-11-jre-headless  # version 11.0.24+8-1ubuntu3-24.04.1
sudo apt install openjdk-8-jre-headless   # version 8u422-b05-1-24.04
sudo apt install openjdk-19-jre-headless  # version 19.0.2+7-4
sudo apt install openjdk-20-jre-headless  # version 20.0.2+9-1
sudo apt install openjdk-22-jre-headless  # version 22-22ea-1
```

gcc –version



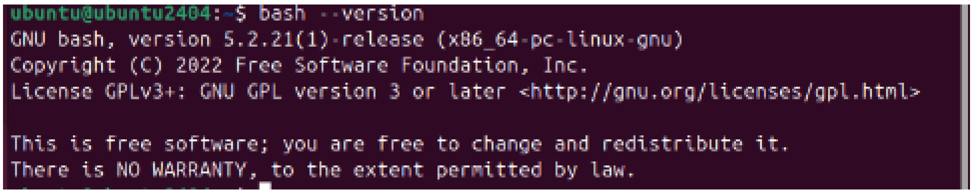
```
ubuntu@ubuntu2404:~$ gcc --version
Command 'gcc' not found, but can be installed with:
sudo apt install gcc
```

python3 –version



```
ubuntu@ubuntu2404:~$ python3 --version
Python 3.12.3
```

bash --version



```
ubuntu@ubuntu2404:~$ 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.
```

Assignment 4.3: Compile

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

Which source code files are compiled into machine code and then directly executable by a processor?

Which source code files are compiled to byte code?

Which source code files are interpreted by an interpreter?

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

How do I run a Java program?

How do I run a Python program?

How do I run a C program?

How do I run a Bash script?

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

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?

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.

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

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

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.

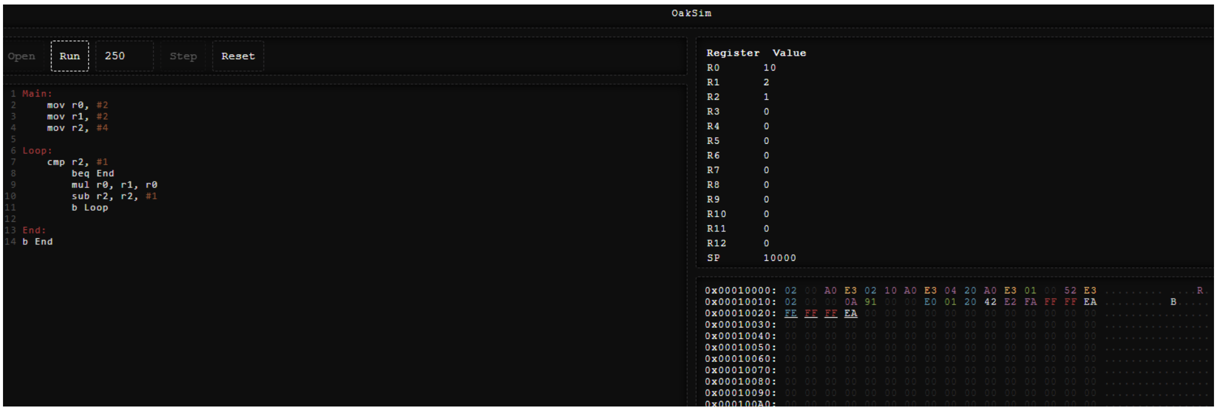
Bonus point assignment – week 4

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, #2

    mov r1, #2

    mov r2, #4


Loop:

    cmp r2, #1

        beq End

        mul r0, r1, r0

        sub r2, r2, #1

        b Loop


End:

b End
```

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

Screenshot of the completed code here.



```
Main:
    mov r0, #2
    mov r1, #2
    mov r2, #4

Loop:
    cmp r2, #1
        beq End
        mul r0, r1, r0
        sub r2, r2, #1
        b Loop

End:
b End
```

```
Register  Value
R0        10
R1        2
R2        1
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
R11       0
R12       0
SP        10000
```

```
0x00010000: 02 00 A0 E3 02 10 A0 E3 04 20 A0 E3 01 00 52 E3 ..............R.
0x00010010: 02 00 00 0A 91 00 00 E0 01 20 42 E2 FA FF FF EA ............B...
0x00010020: FE FF FF EA 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

2^4 = 16   16 is in het hexadecimal 10.
Dus het resultaat klopt.

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