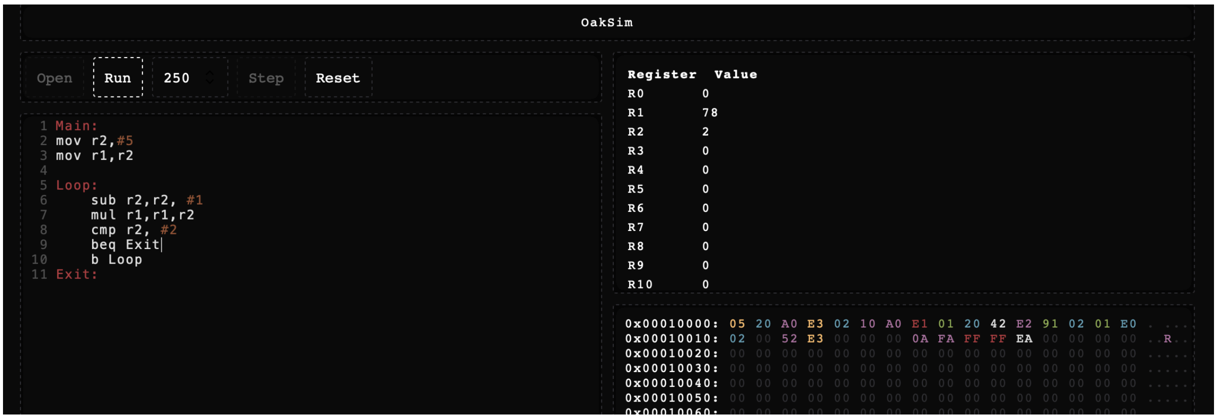
Template Week 4 – Software

Student number:

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
mov r2,#5
mov r1,r2

Loop:
    sub r2,r2, #1
    mul r1,r1,r2
    cmp r2, #2
    beq Exit
    b Loop
Exit:
```

```
Register  Value
R0        0
R1        78
R2        2
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
```

```
0x00010000: 05 20 A0 E3 02 10 A0 E1 01 20 42 E2 91 02 01 E0 . ....
0x00010010: 02 00 52 E3 00 00 00 0A FA FF FF EA 00 00 00 00 ..R..
0x00010020: 00 00 00 00 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
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

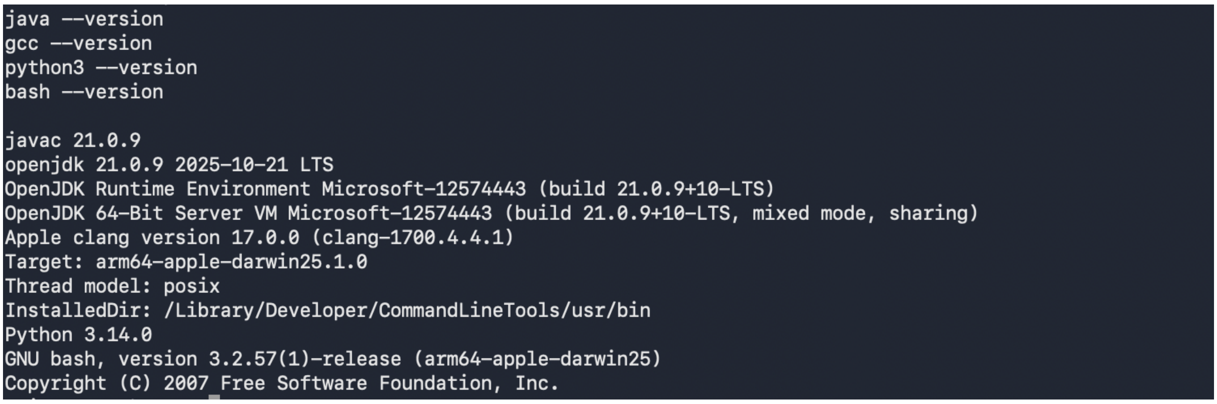
javac --version

java --version

gcc --version

python3 --version

bash --version



```
java --version
gcc --version
python3 --version
bash --version

javac 21.0.9
openjdk 21.0.9 2025-10-21 LTS
OpenJDK Runtime Environment Microsoft-12574443 (build 21.0.9+10-LTS)
OpenJDK 64-Bit Server VM Microsoft-12574443 (build 21.0.9+10-LTS, mixed mode, sharing)
Apple clang version 17.0.0 (clang-1700.4.4.1)
Target: arm64-apple-darwin25.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
Python 3.14.0
GNU bash, version 3.2.57(1)-release (arm64-apple-darwin25)
Copyright (C) 2007 Free Software Foundation, Inc.
```

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?

Python only

Which source code files are compiled to byte code?

Fibonacci.java

Which source code files are interpreted by an interpreter?

Fib.py and fib.sh

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

C program

How do I run a Java program?

Java file

How do I run a Python program?

Python file.py

How do I run a C program?

./file

How do I run a Bash script?

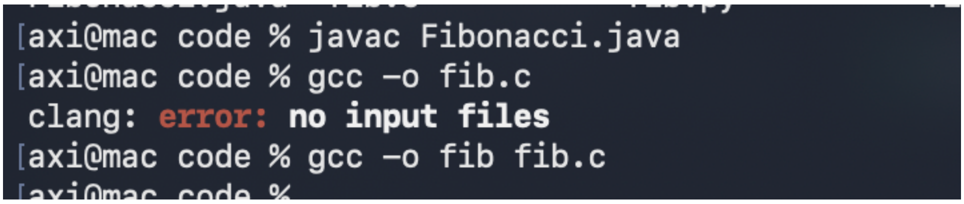
./file.sh

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

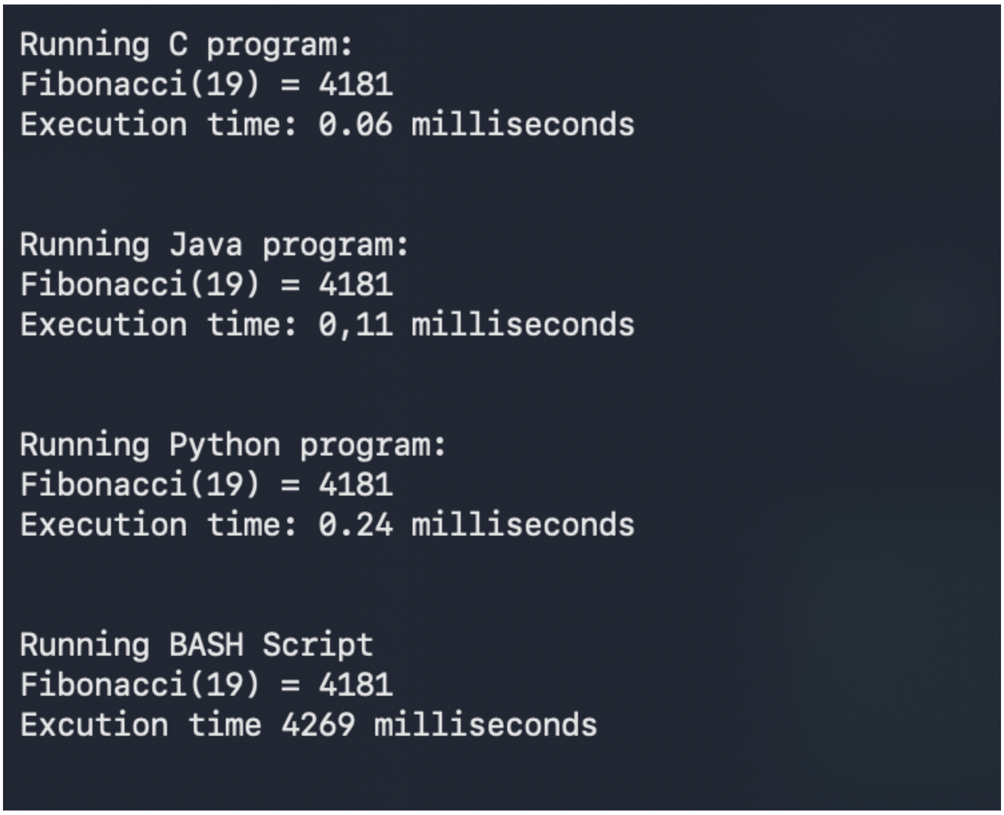
C and java

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?



```
[axi@mac code % javac Fibonacci.java
[axi@mac code % gcc -o fib.c
 clang: error: no input files
[axi@mac code % gcc -o fib fib.c
[axi@mac code %
```



```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.06 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0,11 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.24 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 4269 milliseconds
```

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.

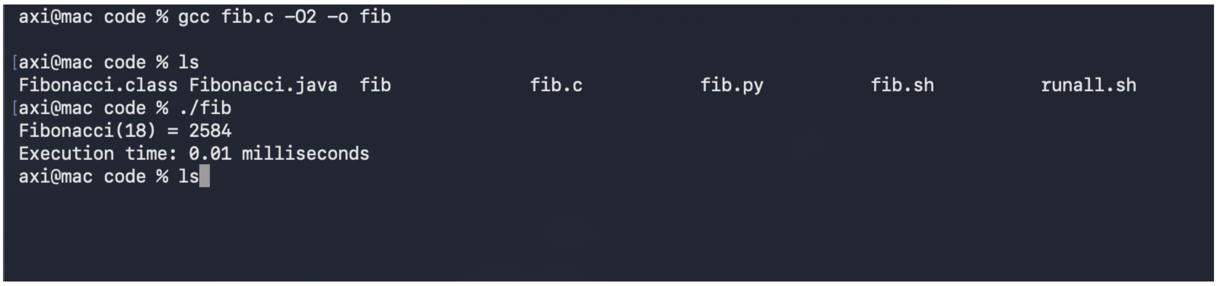
-O1 basic optimization

-O2 good optimization

-O3 aggressive optimization (faster, enables more inlining, vectorization, loop optimizations)

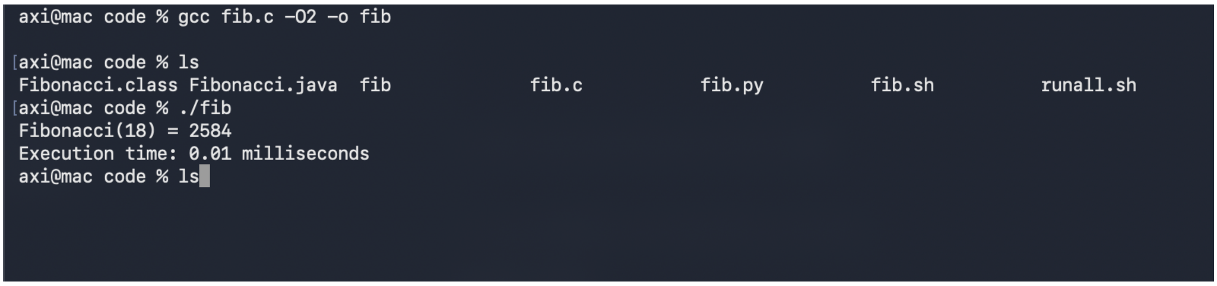
-Ofast even more aggressive, breaks strict standards but often fastest

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



```
axi@mac code % gcc fib.c -O2 -o fib

[axi@mac code % ls
Fibonacci.class Fibonacci.java  fib             fib.c           fib.py          fib.sh          runall.sh
[axi@mac code % ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
axi@mac code % ls
```

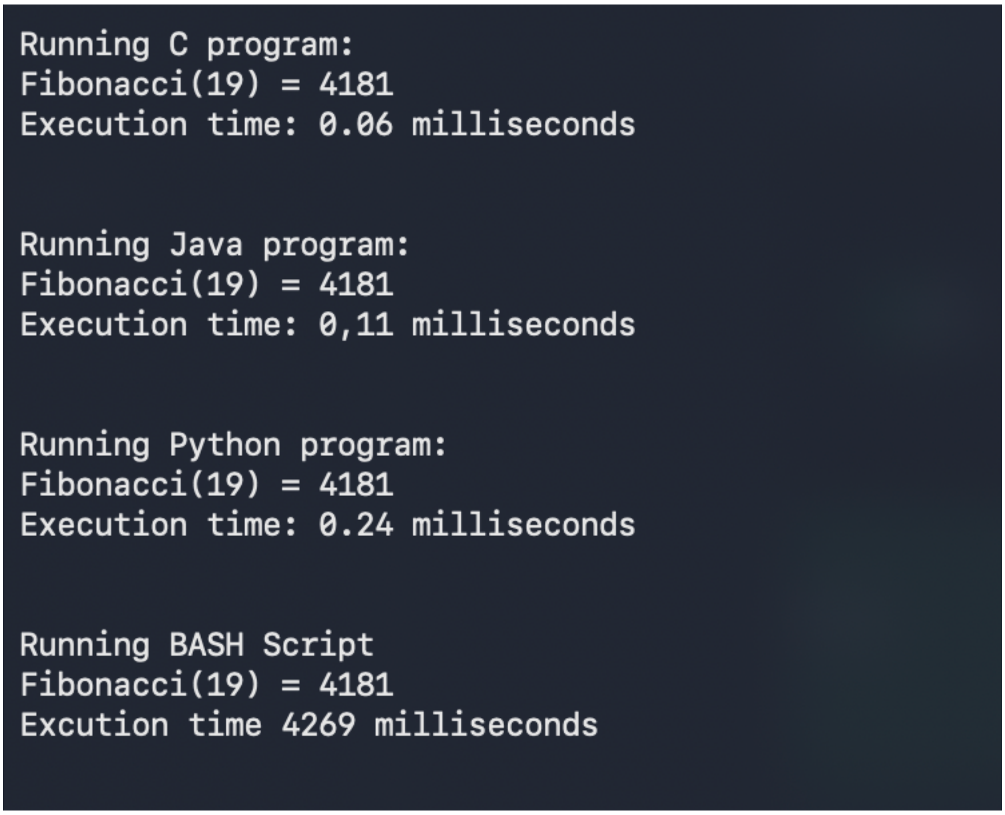
c) Run the newly compiled program. Is it true that it now performs the calculation faster?



```
axi@mac code % gcc fib.c -O2 -o fib

[axi@mac code % ls
Fibonacci.class Fibonacci.java  fib             fib.c           fib.py          fib.sh          runall.sh
[axi@mac code % ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
axi@mac code % ls
```

It is true

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.

e) 

```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.06 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0,11 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.24 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 4269 milliseconds
```

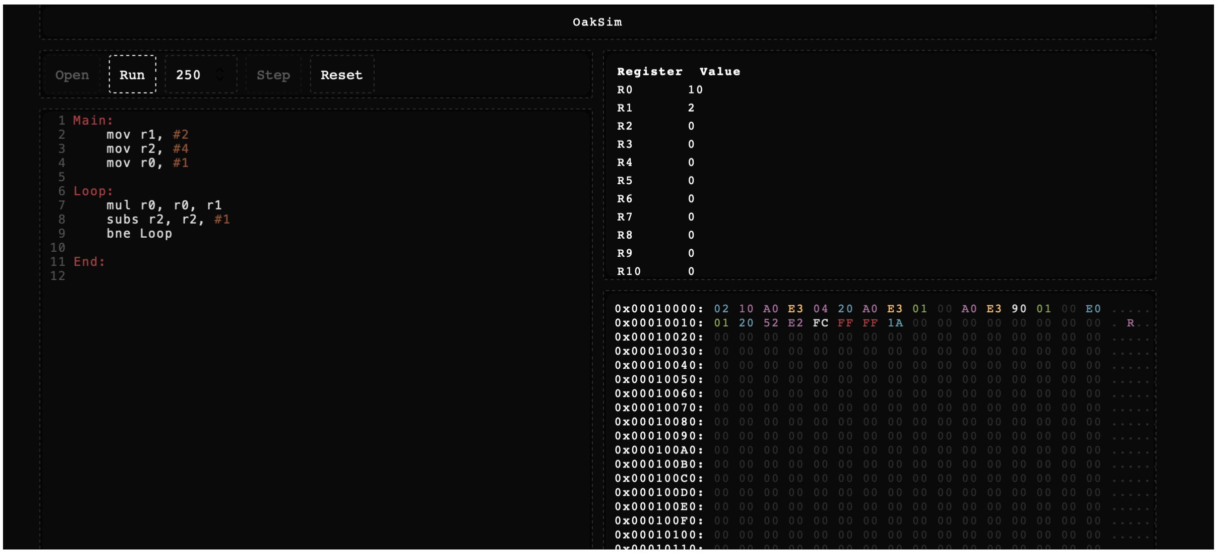
Assignment 4.5: More ARM Assembly

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.

Screenshot of the completed code here.



```
Main:
    mov r1, #2
    mov r2, #4
    mov r0, #1

Loop:
    mul r0, r0, r1
    subs r2, r2, #1
    bne Loop

End:
```

```
Register  Value
R0        10
R1        2
R2        0
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
```

```
0x00010000: 02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 90 01 00 E0 .....
0x00010010: 01 20 52 E2 FC FF FF 1A 00 00 00 00 00 00 00 00 . R..
0x00010020: 00 00 00 00 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 .....
0x000100B0: 00 00 00 00 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 00 00 00 00 .....
0x000100D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x000100E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x000100F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00010100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00010110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

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