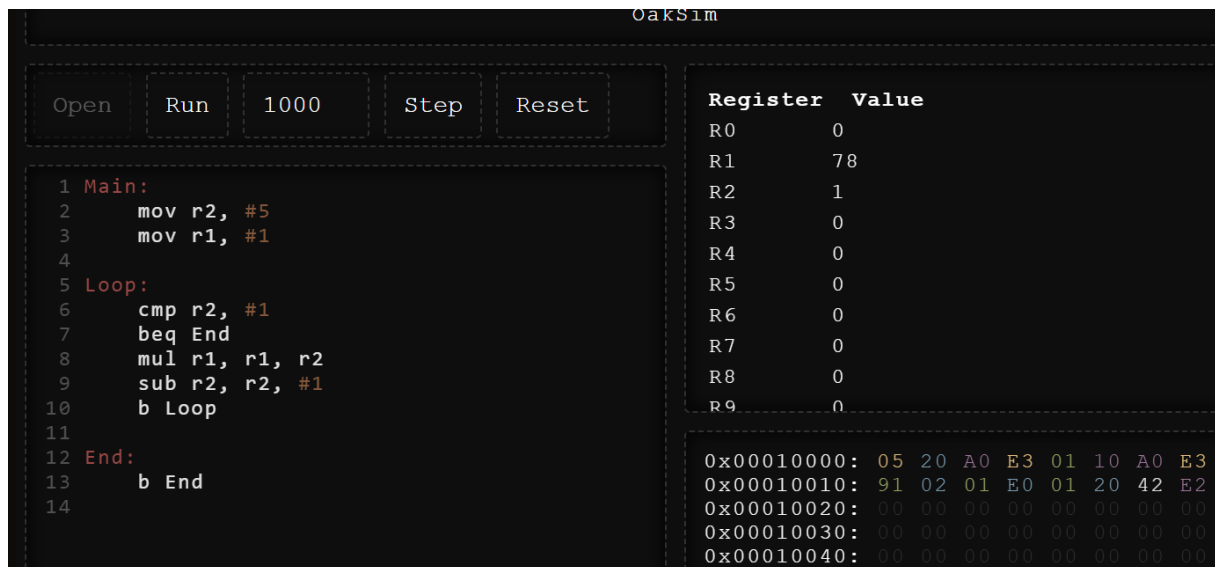


# Template Week 4 – Software

Student number:568261

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



The screenshot shows the OakSim ARM simulator interface. At the top, there are control buttons: Open, Run, 1000, Step, and Reset. The main area is divided into two panels. The left panel displays assembly code for a factorial calculation, and the right panel shows the current state of the registers and memory.

```
1 Main:
2     mov r2, #5
3     mov r1, #1
4
5 Loop:
6     cmp r2, #1
7     beq End
8     mul r1, r1, r2
9     sub r2, r2, #1
10    b Loop
11
12 End:
13    b End
14
```

Register	Value
R0	0
R1	78
R2	1
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0

Address	Value
0x00010000:	05 20 A0 E3 01 10 A0 E3
0x00010010:	91 02 01 E0 01 20 42 E2
0x00010020:	00 00 00 00 00 00 00 00
0x00010030:	00 00 00 00 00 00 00 00
0x00010040:	00 00 00 00 00 00 00 00

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

`javac --version`

```
omer@ubuntu568261:~$ javac --version
javac 21.0.9
```

`java --version`

```
omer@ubuntu568261:~$ 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, s
omer@ubuntu568261:~$
```

`gcc --version`

```
omer@ubuntu568261:~$ 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.
omer@ubuntu568261:~$
```

`python3 --version`

```
omer@ubuntu568261:~$ python3 --version
Python 3.12.3
```

`bash --version`

```
omer@ubuntu568261:~$ bash --version~
bash: --version: invalid option
Usage:  bash [GNU long option] [option] ...
        bash [GNU long option] [option] script-file ...
GNU long options:
  --debug
  --debugger
  --dump-po-strings
  --dump-strings
  --help
  --init-file
  --login
  --noediting
  --noprofile
  --norc
  --posix
  --pretty-print
  --rcfile
  --restricted
  --verbose
  --version
Shell options:
  -ilrsD or -c command or -O shopt_option          (invocation only)
  -abefhkmnptuvxBCEHPT or -o option
omer@ubuntu568261:~$
```

### 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

How do I run a Java program?

javac Fibonacci.java

java Fibonacci

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?

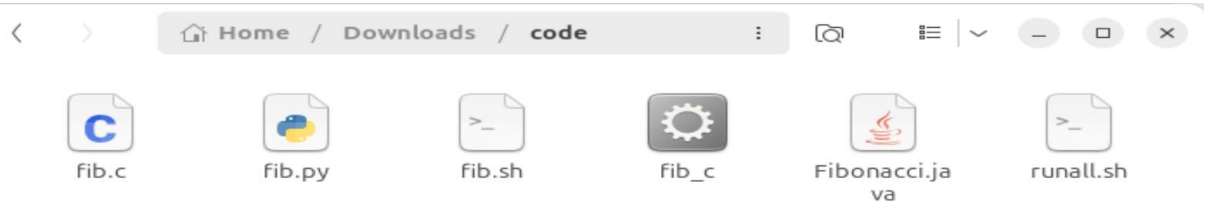
bash fib.sh

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

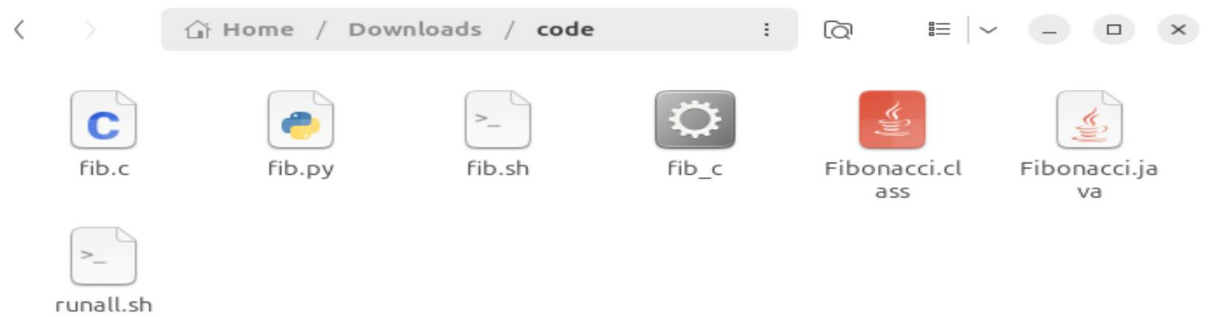
- Fibonacci.java → Fibonacci.class
- fib.c → fib (of fib.exe)

Take relevant screenshots of the following commands:

- Compile the source files where necessary



- Make them executable



- Run them

```
omer@ubuntu568261:~/Downloads/code$ ./fib_c
Fibonacci(18) = 2584
Execution time: 0.05 milliseconds
omer@ubuntu568261:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.85 milliseconds
omer@ubuntu568261:~/Downloads/code$ python3 fib.py
\Fibonacci(18) = 2584
Execution time: 1.14 milliseconds
omer@ubuntu568261:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Execution time 15763 milliseconds
omer@ubuntu568261:~/Downloads/code$
```

- Which (compiled) source code file performs the calculation the fastest?

Fib.C

## Assignment 4.4: Optimize

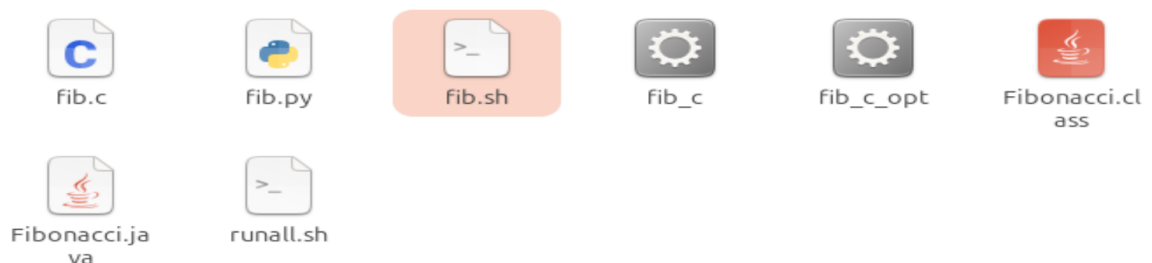
Take relevant screenshots of the following commands:

- 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.

```

GCC(1)                                GNU                                GCC(1)
NAME
    gcc - GNU project C and C++ compiler
SYNOPSIS
    gcc [-c|-S|-E] [-std=standard]
        [-g] [-pg] [-Olevel]
        [-Wwarn...] [-Wpedantic]
        [-Idir...] [-Ldir...]
        [-Dmacro[=defn]...] [-Umacro]
        [-foption...] [-mmachine-option...]
        [-o outfile] [@file] infile...
    Only the most useful options are listed here; see below for the
    remainder.  g++ accepts mostly the same options as gcc.
DESCRIPTION
    When you invoke GCC, it normally does preprocessing, compilation,
    assembly and linking.  The "overall options" allow you to stop this
    process at an intermediate stage.  For example, the -c option says not
    to run the linker.  Then the output consists of object files output by
    the assembler.
Manual page gcc(1) line 1 (press h for help or q to quit)
```

- 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?

```

omer@ubuntu568261:~/Downloads/code$ ./fib_c_opt
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
```

Yes it runs 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.

```
Running Java program:
Fibonacci(19) = 4181
Execution time: 1.66 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 1.10 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 27379 milliseconds

Running C program
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
Running Java program
Fibonacci(18) = 2584
Execution time: 0.89 milliseconds
Running Python program
Fibonacci(18) = 2584
Execution time: 0.82 milliseconds
Running Bash script
Fibonacci(18) = 2584
Execution time 17407 milliseconds
omer@ubuntu568261:~/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:

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

Screenshot of the completed code here.

Main:

```
mov r1, #2    @ base = 2
mov r2, #4    @ exponent = 4
mov r0, #1    @ result = 1
```

Loop:

```
mul r0, r0, r1 @ result = result * 2
subs r2, r2, #1 @ exponent--
bne Loop      @ repeat if exponent != 0
```

End:

Ready? Save this file and export it as a pdf file with the name: [week4.pdf](#)