

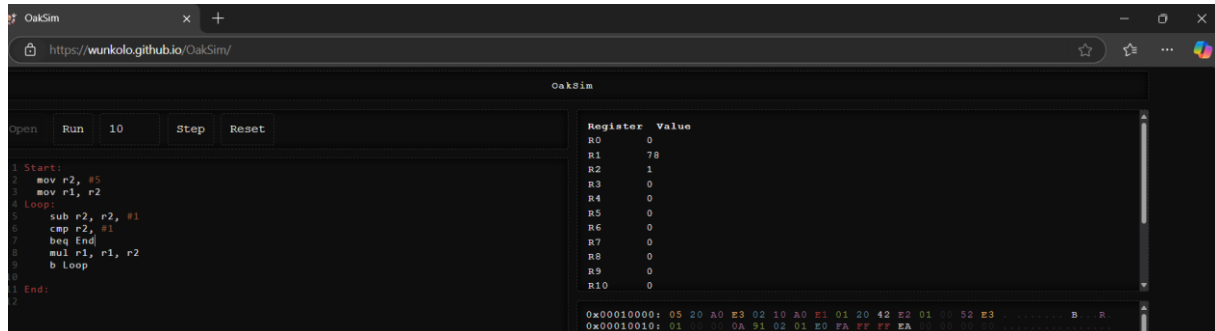
# Template Week 4 – Software

Student number: 568209 eilyad

Used chatcpt only for better English writing and getting some information about assignment 4.3

## Assignment 4.1: ARM assembly

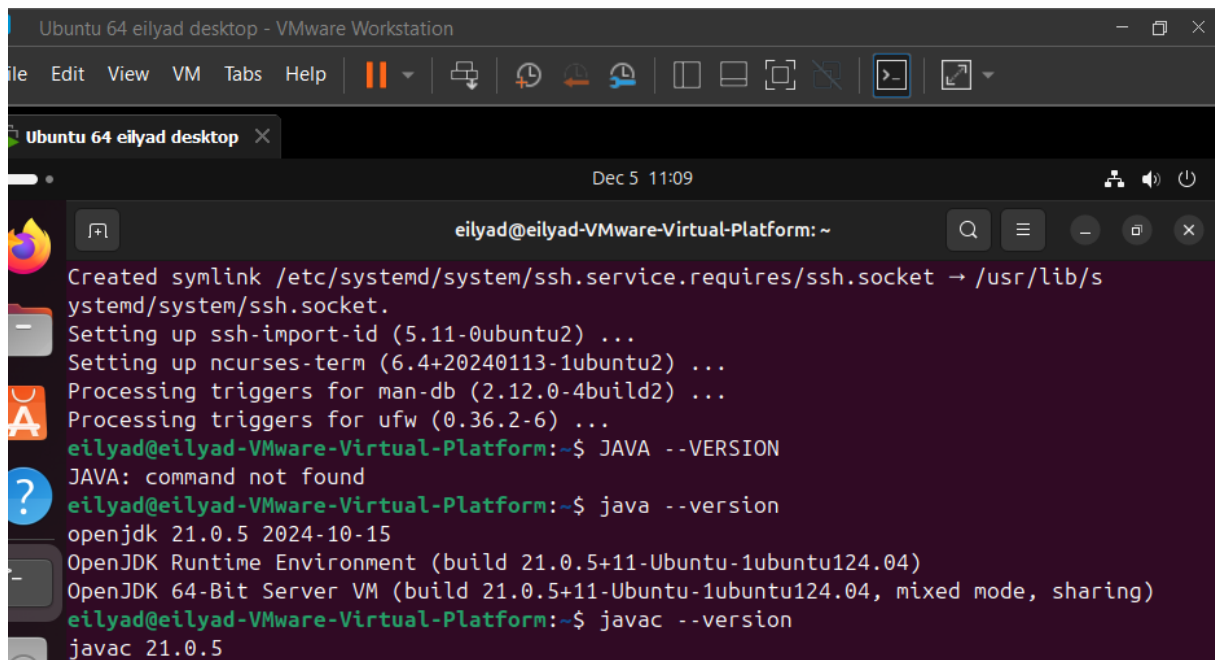
Screenshot of working assembly code of factorial calculation:



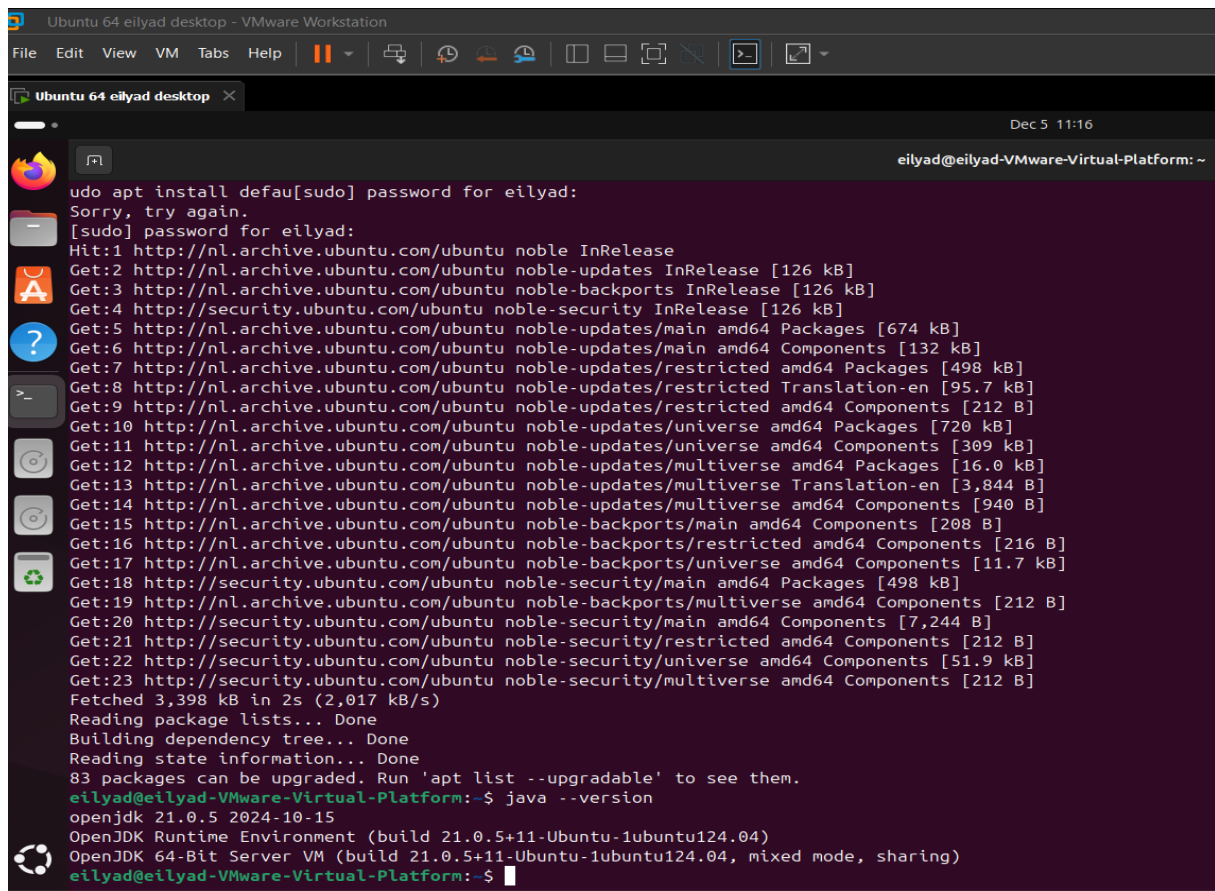
## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

`javac --version`



java -version



```
Ubuntu 64 eilyad desktop - VMware Workstation
File Edit View VM Tabs Help
Ubuntu 64 eilyad desktop x
Dec 5 11:16
eilyad@eilyad-VMware-Virtual-Platform: ~
udo apt install defau[sudo] password for eilyad:
Sorry, try again.
[sudo] password for eilyad:
Hit:1 http://nl.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://nl.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://nl.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:5 http://nl.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [674 kB]
Get:6 http://nl.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [132 kB]
Get:7 http://nl.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [498 kB]
Get:8 http://nl.archive.ubuntu.com/ubuntu noble-updates/restricted Translation-en [95.7 kB]
Get:9 http://nl.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
Get:10 http://nl.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [720 kB]
Get:11 http://nl.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [309 kB]
Get:12 http://nl.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [16.0 kB]
Get:13 http://nl.archive.ubuntu.com/ubuntu noble-updates/multiverse Translation-en [3,844 B]
Get:14 http://nl.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [940 B]
Get:15 http://nl.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [208 B]
Get:16 http://nl.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
Get:17 http://nl.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [11.7 kB]
Get:18 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [498 kB]
Get:19 http://nl.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Get:20 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [7,244 B]
Get:21 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:22 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [51.9 kB]
Get:23 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
Fetched 3,398 kB in 2s (2,017 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
83 packages can be upgraded. Run 'apt list --upgradable' to see them.
eilyad@eilyad-VMware-Virtual-Platform:~$ java -version
openjdk 21.0.5 2024-10-15
OpenJDK Runtime Environment (build 21.0.5+11-Ubuntu-1ubuntu124.04)
OpenJDK 64-Bit Server VM (build 21.0.5+11-Ubuntu-1ubuntu124.04, mixed mode, sharing)
eilyad@eilyad-VMware-Virtual-Platform:~$
```

gcc -version



### Assignment 4.3: Compile

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

Fibonacci.java (compiled to bytecode).

Fib.c (compiled to machine code).

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

fib.c is compiled into machine code using gcc.

Which source code files are compiled to byte code?

Fibonacci.java is compiled into bytecode (Fibonacci.Class) by the javac command.

Which source code files are interpreted by an interpreter?

Python programs (e.g., fib.py) are interpreted by the Python interpreter (python3). Bash scripts (e.g., fib.sh) are interpreted by the Bash shell

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

C program (compiled to machine code) is expected to perform calculations the fastest because it runs directly on the hardware without any intermediate layer like the JVM or an interpreter

How do I run a Java program?

Compile: `javac Fibonacci.java`

Run: `java Fibonacci`

How do I run a Python program?

Run the Python script directly using: `python3 fib.py`

How do I run a C program?

Compile: `gcc -o fib Fib.c`

Run: `./fib`

How do I run a Bash script?

Make the script executable: `chmod +x fib.sh`

Run: `./fib.sh`

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

### **Fibonacci.java**

Compiled into: `Fibonacci.class` (bytecode file for the JVM).

### **Fib.c**

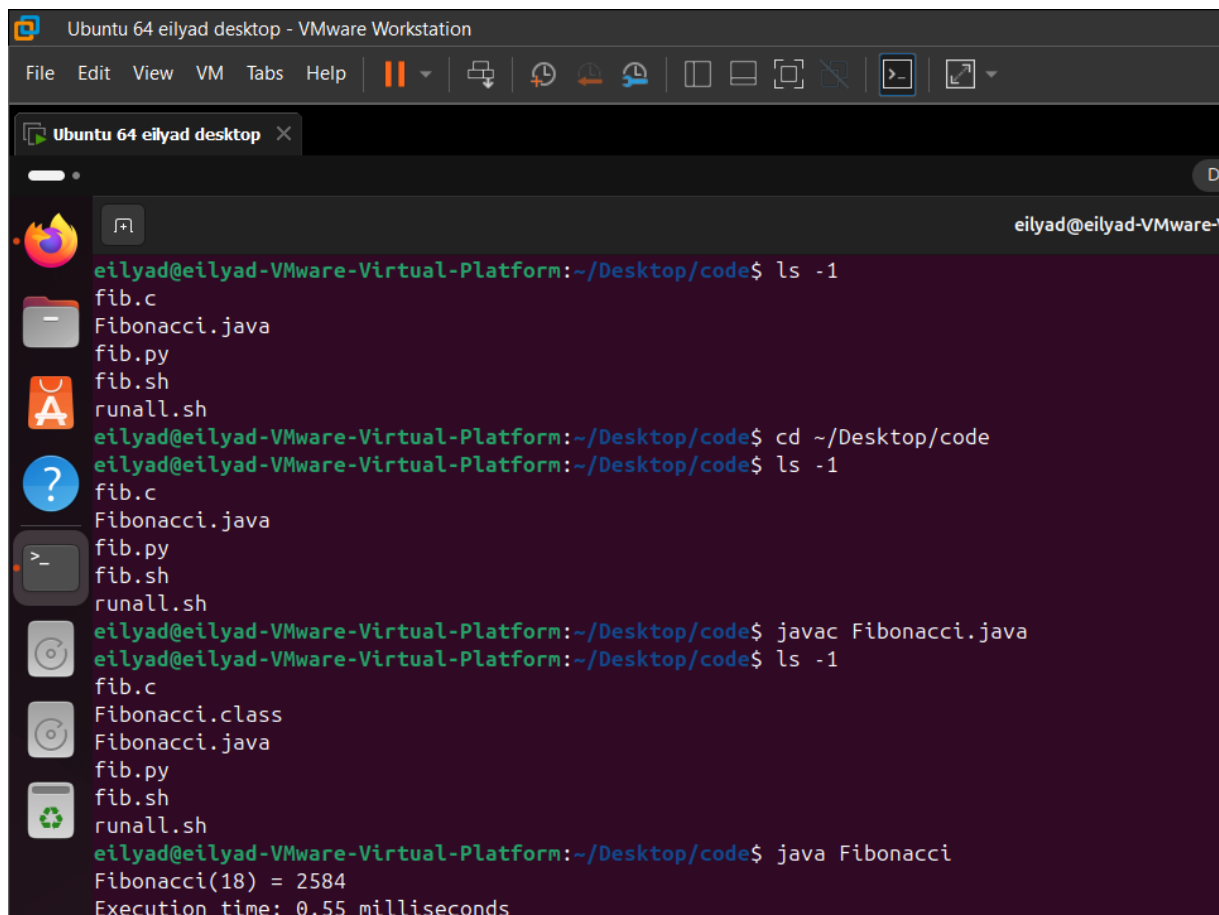
Compiled into: `fib` (machine code executable).

### **Python and Bash scripts**

No new files are created since they are interpreted directly without compilation.

Take relevant screenshots of the following commands:

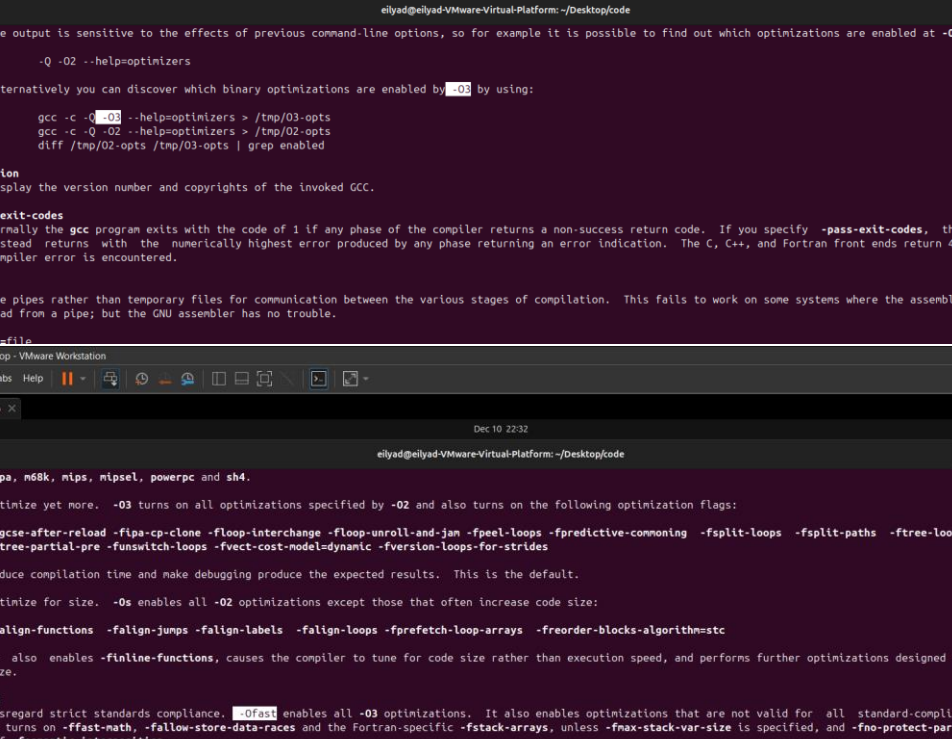
- Compile the source files where necessary
- 
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?



```
Ubuntu 64 eilyad desktop - VMware Workstation
File Edit View VM Tabs Help
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ls -l
fib.c
Fibonacci.java
fib.py
fib.sh
runall.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ cd ~/Desktop/code
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ls -l
fib.c
Fibonacci.java
fib.py
fib.sh
runall.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ javac Fibonacci.java
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ls -l
fib.c
Fibonacci.class
Fibonacci.java
fib.py
fib.sh
runall.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.55 milliseconds
```

```
Ubuntu 64 eilyad desktop - VMware Workstation
File Edit View VM Tabs Help
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -o fib fib.c
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ls -l
-rwxr-xr-x 1 eilyad eilyad 12288 Feb 10 10:10 fib
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 fib.c
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 Fibonacci.class
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 Fibonacci.java
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 fib.py
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 fib.sh
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 runall.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.04 milliseconds
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.37 milliseconds
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ chmod +x fib.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ls -l
-rwxr-xr-x 1 eilyad eilyad 12288 Feb 10 10:10 fib
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 fib.c
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 Fibonacci.class
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 Fibonacci.java
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 fib.py
-rwxr-xr-x 1 eilyad eilyad 12288 Feb 10 10:10 fib.sh
-rw-r--r-- 1 eilyad eilyad 12288 Feb 10 10:10 runall.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 20577 milliseconds
```

Take relevant screenshots of the following commands:

- 
- ```

Ubuntu 64 eliyad desktop - VMware Workstation
File Edit View VM Tabs Help
Dec 10 22:30
eliyad@eliyad-VMware-Virtual-Platform: ~/Desktop/code

The output is sensitive to the effects of previous command-line options, so for example it is possible to find out which optimizations are enabled at -O2 by using:

-Q -O2 --help=optimizers

Alternatively you can discover which binary optimizations are enabled by -O3 by using:

gcc -c -O -O3 --help=optimizers > /tmp/O3-opts
gcc -c -O -O2 --help=optimizers > /tmp/O2-opts
diff /tmp/O2-opts /tmp/O3-opts | grep enabled

--version
Display the version number and copyrights of the invoked GCC.

-pass-exit-codes
Normally the gcc program exits with the code of 1 if any phase of the compiler returns a non-success return code. If you specify -pass-exit-codes, the gcc program instead returns with the numerically highest error produced by any phase returning an error indication. The C, C++, and Fortran front ends return 4 if an internal compiler error is encountered.

-pipe
Use pipes rather than temporary files for communication between the various stages of compilation. This fails to work on some systems where the assembler is unable to read from a pipe; but the GNU assembler has no trouble.

-snooze-file

Ubuntu 64 eliyad desktop - VMware Workstation
File Edit View VM Tabs Help
Dec 10 22:32
eliyad@eliyad-VMware-Virtual-Platform: ~/Desktop/code

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 -fswitch-loops -fvect-cost-model=dynamic -fversion-loops-for-strides

-O0 Reduce compilation time and make debugging produce the expected results. This is the default.

-Os Optimize for size. -Os enables all -O2 optimizations except those that often increase code size:

-falign-functions -falign-jumps -falign-labels -falign-loops -fprefetch-loop-arrays -freorder-blocks-algorithm=stc

It also enables -finline-functions, causes the compiler to tune for code size rather than execution speed, and performs further optimizations designed to reduce code size.

-Ofast
Disregard strict standards compliance. -Ofast enables all -O3 optimizations. It also enables optimizations that are not valid for all standard-compliant programs. It turns on -ffast-math, -fallow-store-data-races and the Fortran-specific -fstack-arrays, unless -fmax-stack-var-size is specified, and -fno-protect-parens. It turns off -fsemantic-interposition.

-Og Optimize debugging experience. -Og should be the optimization level of choice for the standard edit-compile-debug cycle, offering a reasonable level of optimization while maintaining fast compilation and a good debugging experience. It is a better choice than -O0 for producing debuggable code because some compiler passes that collect debug information are disabled at -O0.

Like -O0, -Og completely disables a number of optimization passes so that individual options controlling them have no effect. Otherwise -Og enables all -O1 optimization flags except for those that may interfere with debugging:

```

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



```
Ubuntu 64 eilyad desktop - VMware Workstation
File Edit View VM Tabs Help
Ubuntu 64 eilyad desktop x
Dec 10 22:41
eilyad@eilyad-VMware-Virtual-Platform: ~/Desktop/code
Execution time 464313215 milliseconds
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ man gcc
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ man gcc
[1]+  Stopped                  man gcc
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ man gcc
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ nano Fib.c
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -o fib Fib.c
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/13/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -O3 -o fib_optimized Fib.c
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/13/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ man gcc
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ nano Fib.c
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -o fib Fib.c
./fib
Fibonacci(10) = 55
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -O3 -o fib_optimized Fib.c
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ls
-O3  3  fib  fib.c  Fib.c  Fibonacci.class  Fibonacci.java  fib_optimized  fib.py  fib.sh  'ic optimizations.'  runall.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$
```

- c) Run the newly compiled program. Is it true that it now performs the calculation faster?  
The optimized version uses advanced compiler optimizations, reducing execution time by improving loop efficiency, inlining functions, and simplifying calculations.

```
Ubuntu 64 eilyad desktop - VMware Workstation
File Edit View VM Tabs Help
Ubuntu 64 eilyad desktop x
Dec 10 22:42
eilyad@eilyad-VMware-Virtual-Platform: ~/Desktop/code
collect2: error: ld returned 1 exit status
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -O3 -o fib_optimized Fib.c
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/13/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ man gcc
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ nano Fib.c
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -o fib Fib.c
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/13/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ nano Fib.c
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -o fib Fib.c
./fib
Fibonacci(10) = 55
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ gcc -O3 -o fib_optimized Fib.c
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ ls
-O3  3  fib  fib.c  Fib.c  Fibonacci.class  Fibonacci.java  fib_optimized  fib.py  fib.sh  'ic optimizations.'  runall.sh
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ time ./fib
Fibonacci(10) = 55

real    0m0.004s
user    0m0.003s
sys     0m0.001s
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/code$ time ./fib_optimized
Fibonacci(10) = 55

real    0m0.004s
user    0m0.002s
sys     0m0.002s
eilyad@eilyad-VMware-Virtual-Platform:~/Desktop/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.

C: Runs directly on the hardware as compiled machine code → fastest.

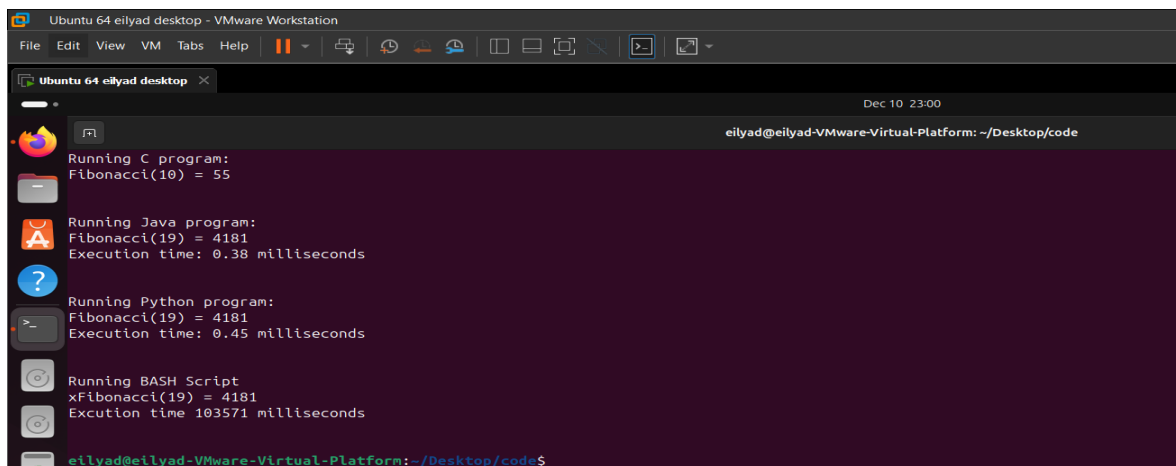
Java: Runs on the Java Virtual Machine (JVM), which adds some overhead.

Python: Interpreted language → Slower than Java or C.

Bash: Designed for scripting, not computation → Slowest.

C is the fastest because the code is compiled into native machine code and runs directly on the CPU.

Optimizations like `-O3` make C even faster by improving how loops and functions are handled.



The screenshot shows a terminal window titled 'Ubuntu 64 eilyad desktop - VMware Workstation'. The terminal output is as follows:

```
Running C program:
Fibonacci(10) = 55

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.38 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.45 milliseconds

Running BASH Script
xFibonacci(19) = 4181
Execution time 103571 milliseconds
```

The prompt at the bottom is `eilyad@eilyad-VMware-Virtual-Platform: ~/Desktop/code$`.

#### 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 r1, #2
```

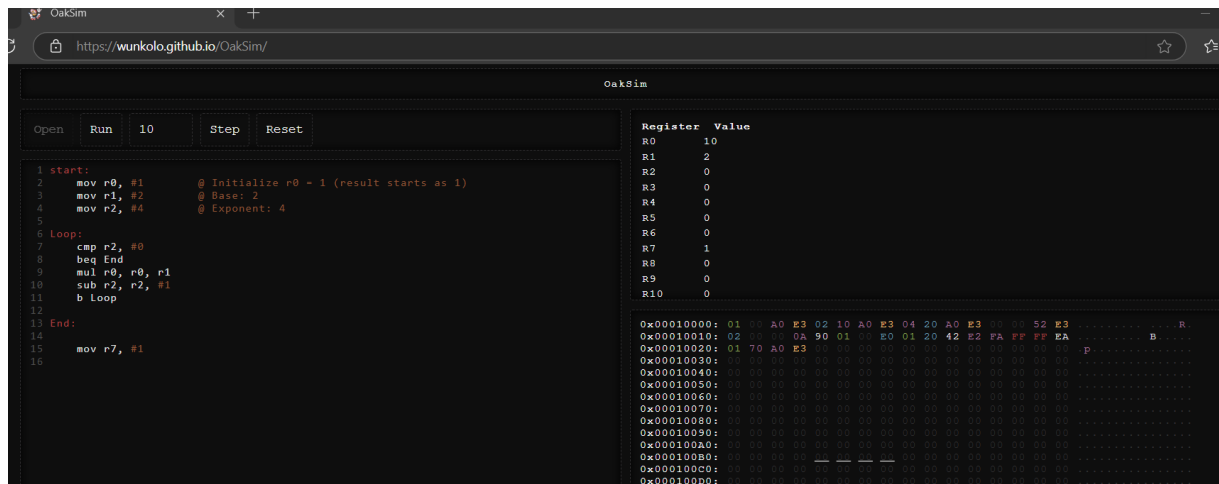
```
mov r2, #4
```

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](#)

(I used ChatGPT to get informations extra and write it in a better english)