

Template Week 4 – Software

Student number: 563634

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

```
1 // Main
2 add r0, r0, r2
3 mul r1, r0, r0
4 sub r2, r1, r0
5 mov r3, r15
6
7 // 563634 Yonatan Amizag
```

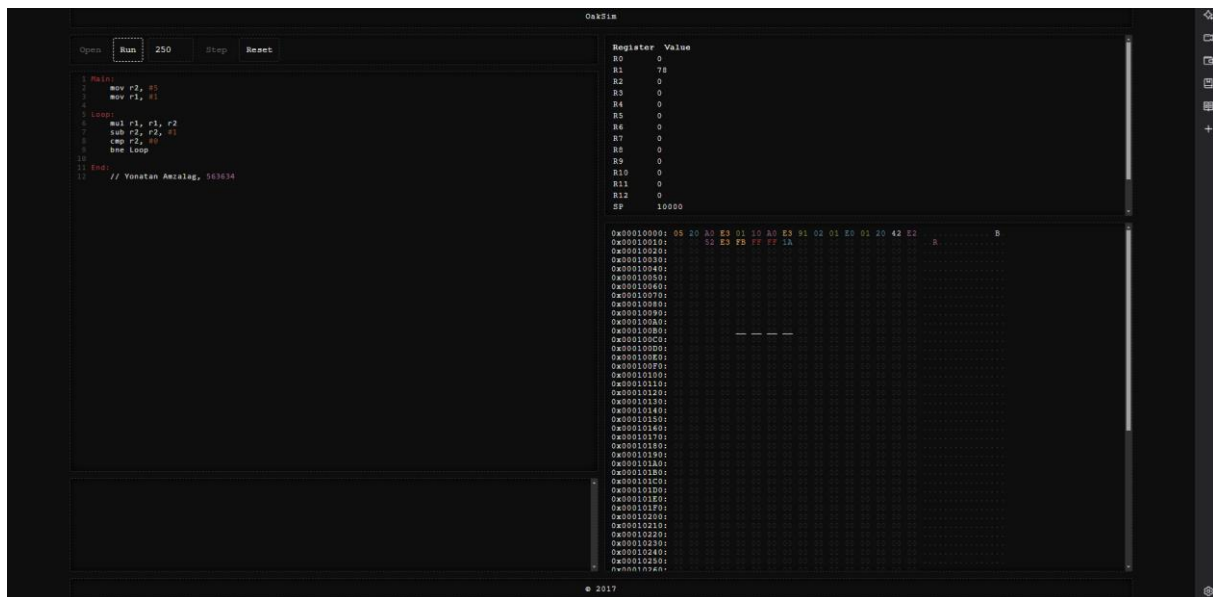
Register Value

Register	Value
R0	2
R1	4
R2	2
R3	4
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
SP	10000

```
1 Loop:
2 add r0, r0, r1
3 mul r1, r0, r0
4 cmp r1, #16
5 bgt Exit
6 b Loop
7 Exit:
8
9 // 563634 Yonatan Amizag
```

Register Value

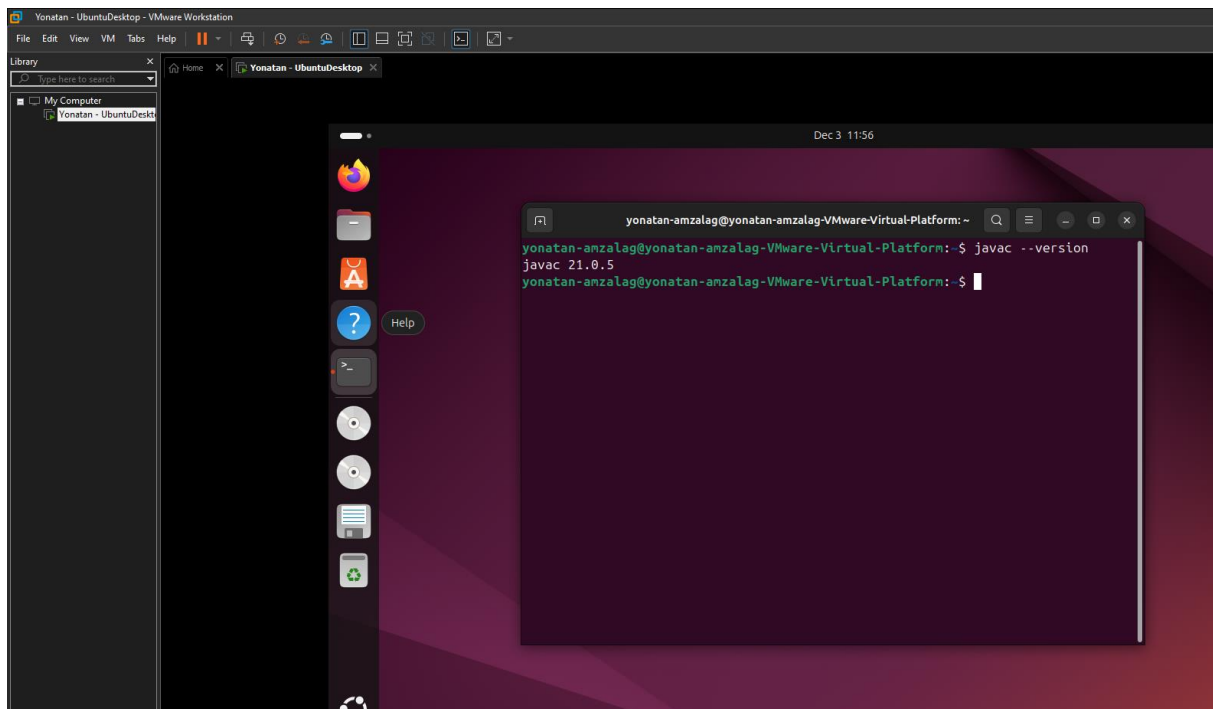
Register	Value
R0	0
R1	16
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
SP	10000



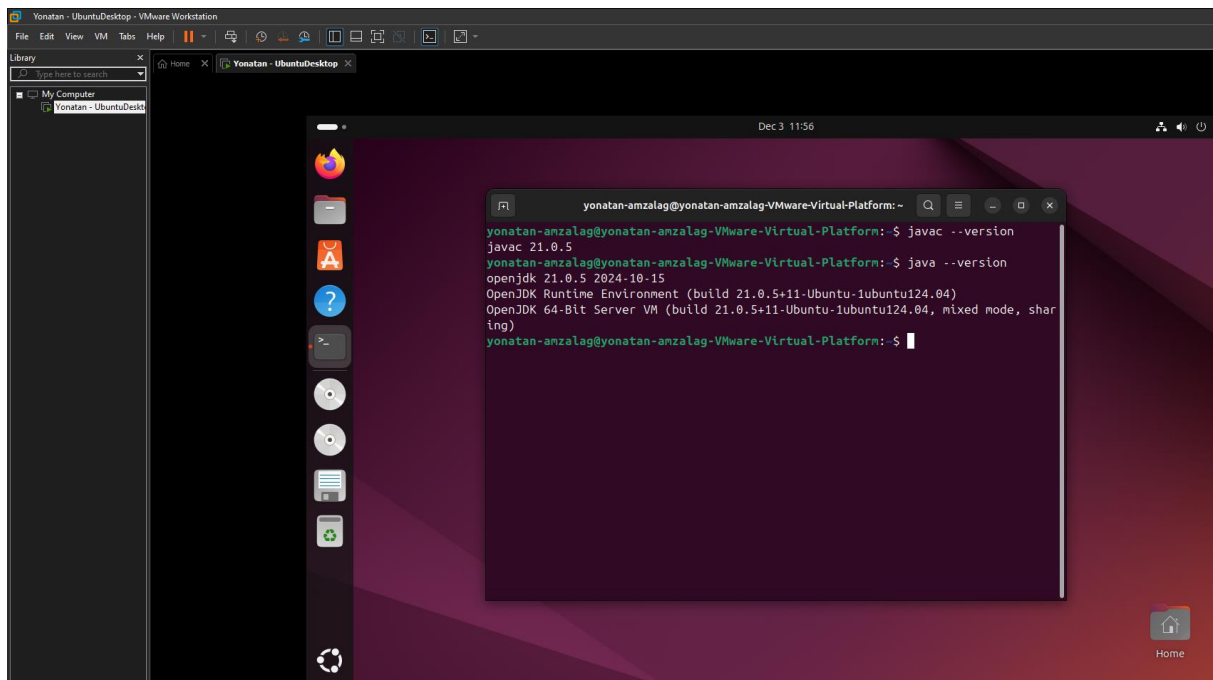
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version



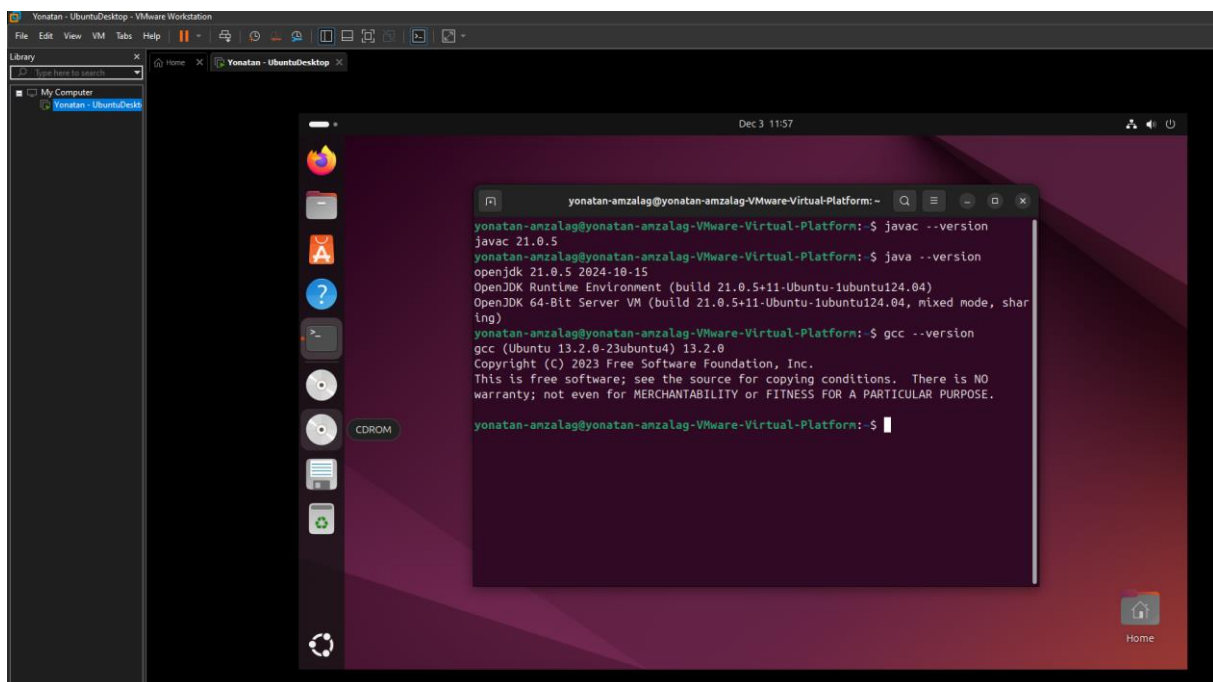
java --version



The screenshot shows a VMware Workstation window titled "Yonatan - UbuntuDesktop - VMware Workstation". Inside the virtual machine, a terminal window is open with the following output:

```
yonatan-amzalag@yonatan-amzalag-VMware-Virtual-Platform: ~  
yonatan-amzalag@yonatan-amzalag-VMware-Virtual-Platform: $ javac --version  
javac 21.0.5  
yonatan-amzalag@yonatan-amzalag-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)  
yonatan-amzalag@yonatan-amzalag-VMware-Virtual-Platform: $
```

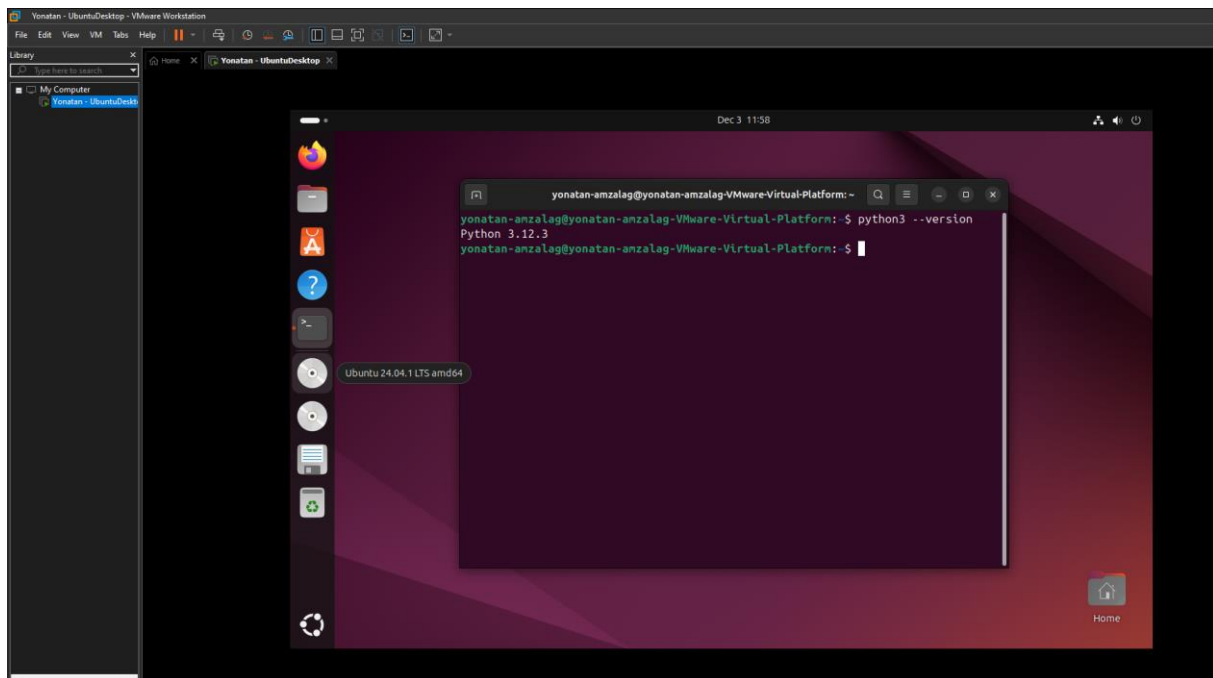
gcc --version



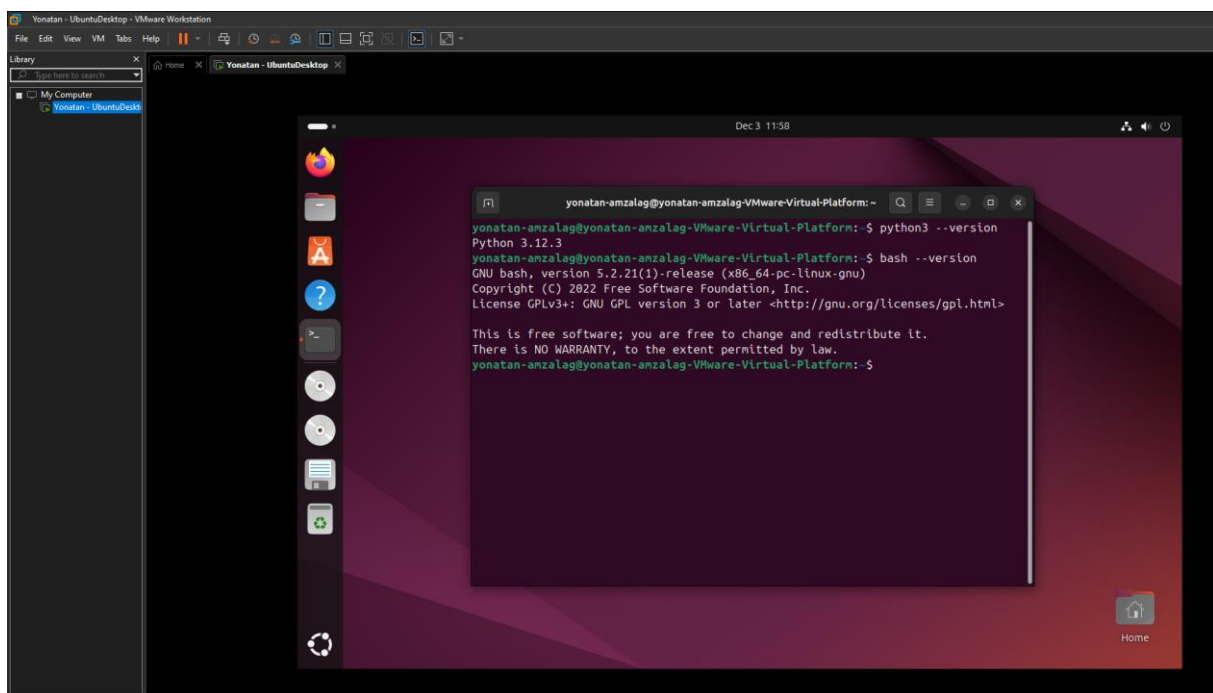
The screenshot shows a VMware Workstation window titled "Yonatan - UbuntuDesktop - VMware Workstation". Inside the virtual machine, a terminal window is open with the following output:

```
yonatan-amzalag@yonatan-amzalag-VMware-Virtual-Platform: ~  
yonatan-amzalag@yonatan-amzalag-VMware-Virtual-Platform: $ javac --version  
javac 21.0.5  
yonatan-amzalag@yonatan-amzalag-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)  
yonatan-amzalag@yonatan-amzalag-VMware-Virtual-Platform: $ gcc --version  
gcc (Ubuntu 13.2.0-23ubuntu4) 13.2.0  
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.  
yonatan-amzalag@yonatan-amzalag-VMware-Virtual-Platform: $
```

python3 --version



bash --version



Assignment 4.3: Compile

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

Fibonacci.java and fib.c

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

The fib.c source code and using a compiler like gcc.

Which source code files are compiled to byte code?

The Fibonacci.java source code, using the compiler javac.

Which source code files are interpreted by an interpreter?

Fib.py interpreted by python3 interpreter, and fib.sh interpreted by Bash shell.

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

Fib.c, it's a compiled C code that is executed instantly by the processor.

How do I run a Java program?

We need to compile the program first, and then run the program using the Java interpreter (javac)

How do I run a Python program?

We use the python3 interpreter.

How do I run a C program?

We can use a compiler like gcc to compile it, and then execute it.

How do I run a Bash script?

Run the script.

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

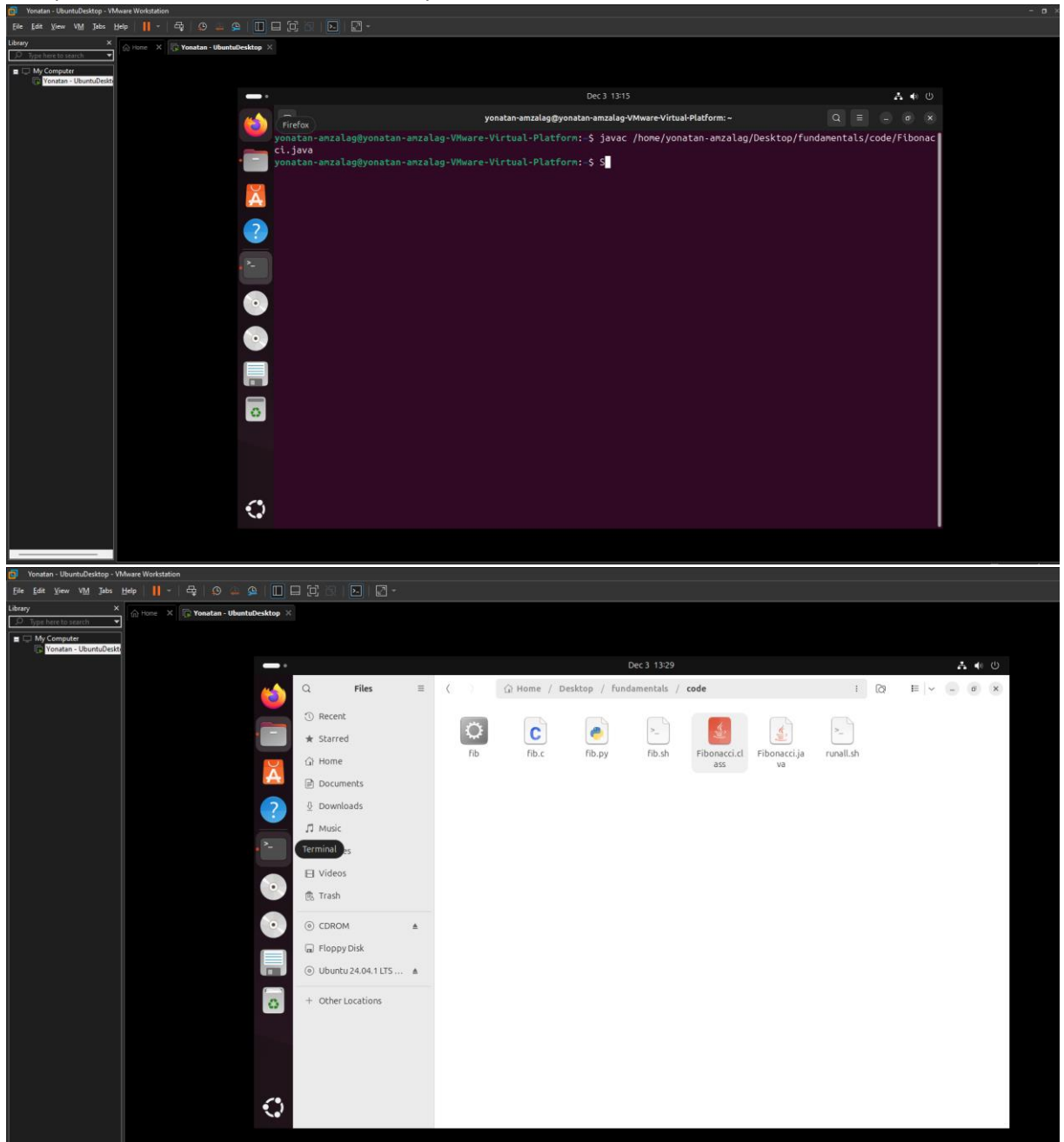
When we run the Fibonacci.java file, a Fibonacci.class file will be created.

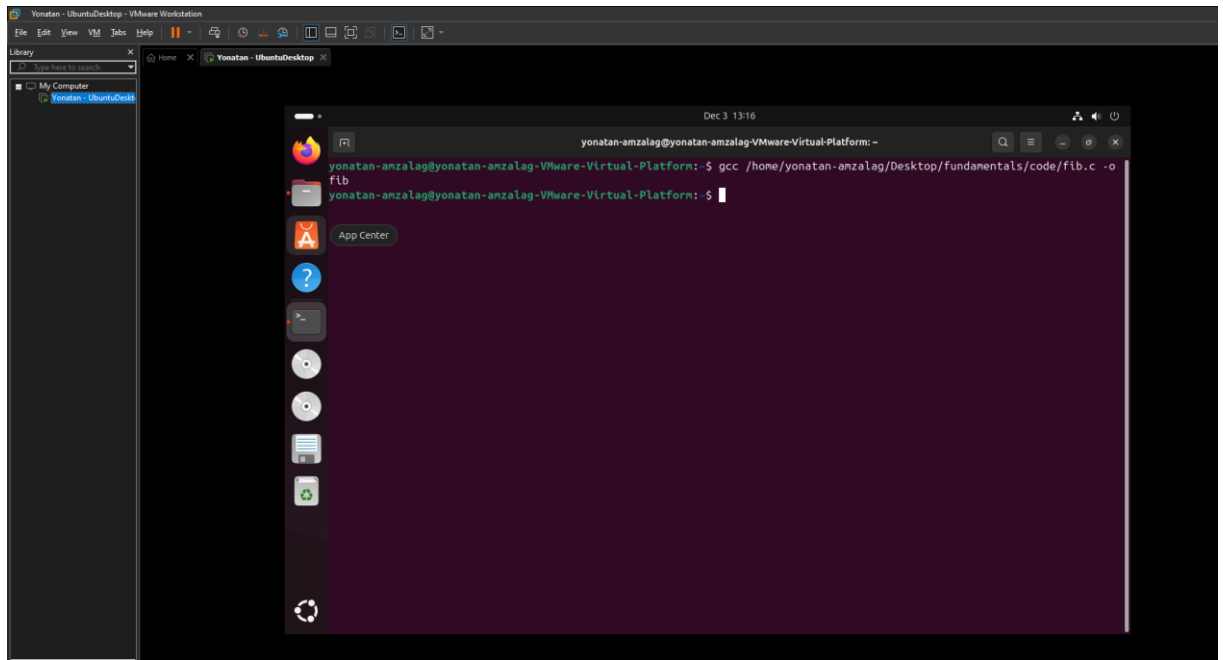
When we run the fib.c file, an executable file will be created.

When we use python and bash, no new file will be created.

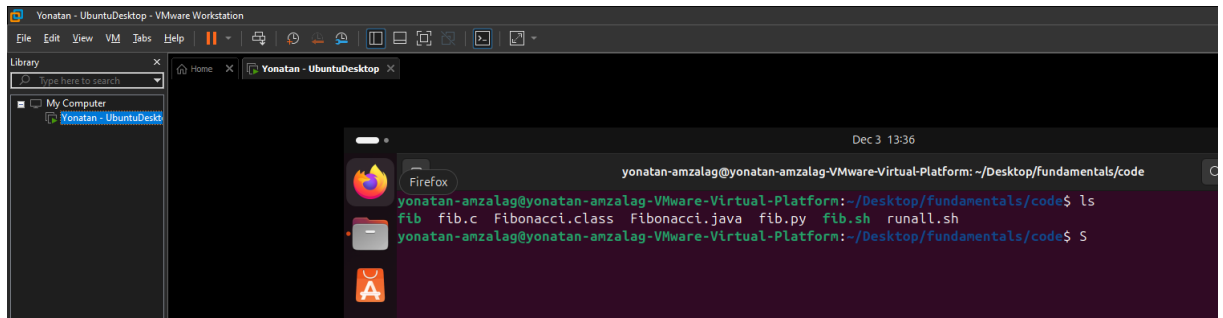
Take relevant screenshots of the following commands:

- Compile the source files where necessary

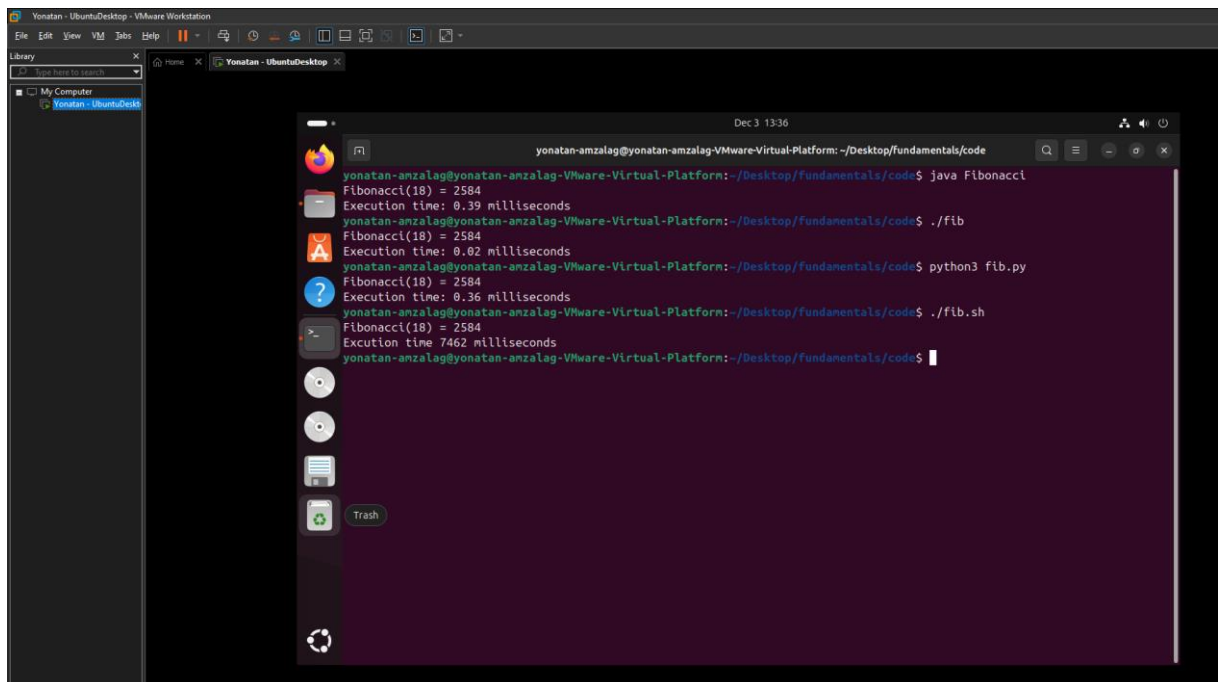




- Make them executable



- Run them

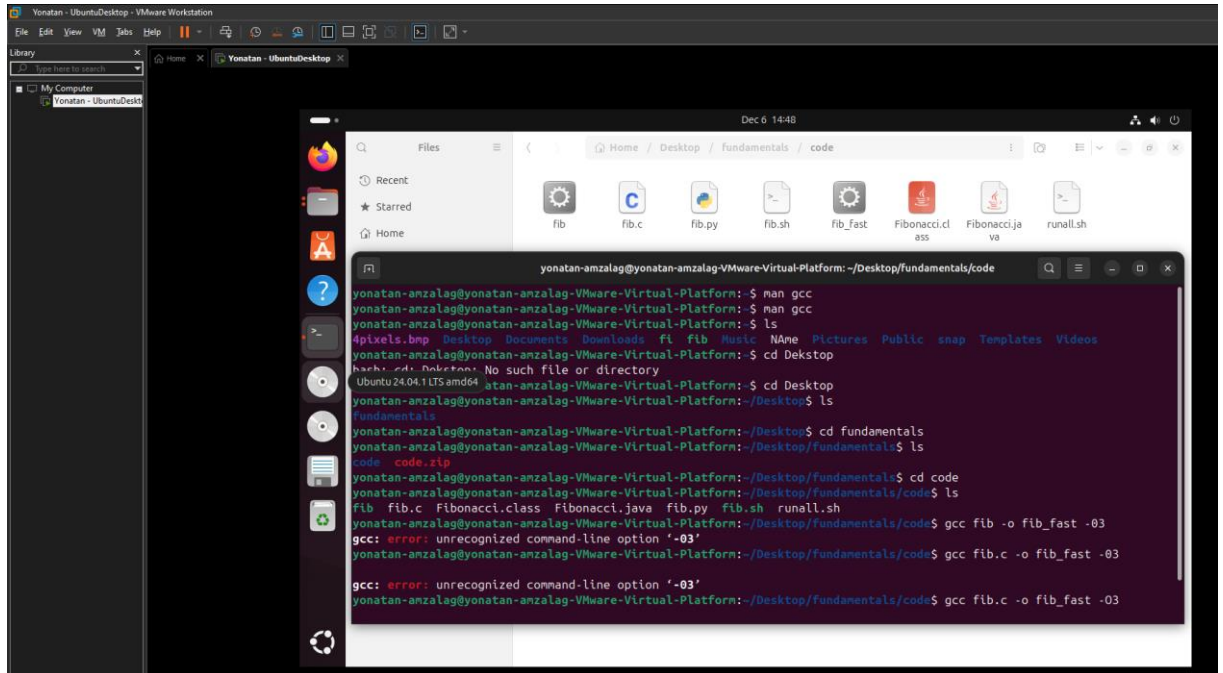


- Which (compiled) source code file performs the calculation the fastest?
The fastest is C, because it's a compiled machine code.

Assignment 4.4: Optimize

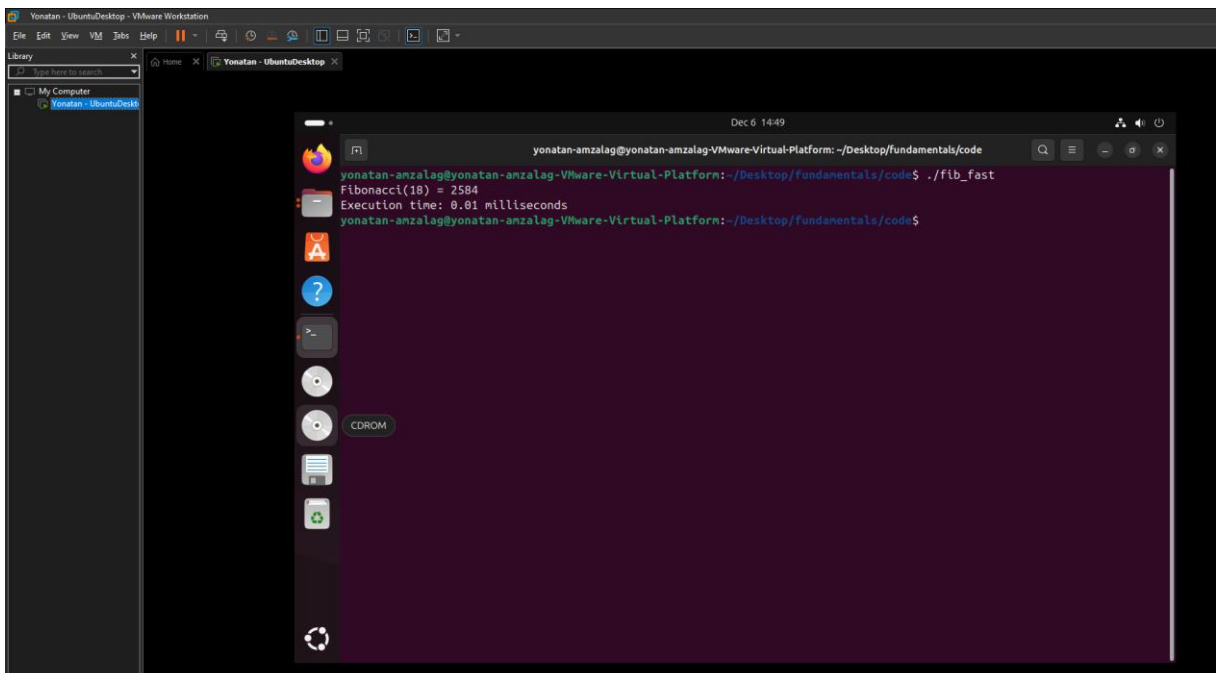
Take relevant screenshots of the following commands:

- 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.



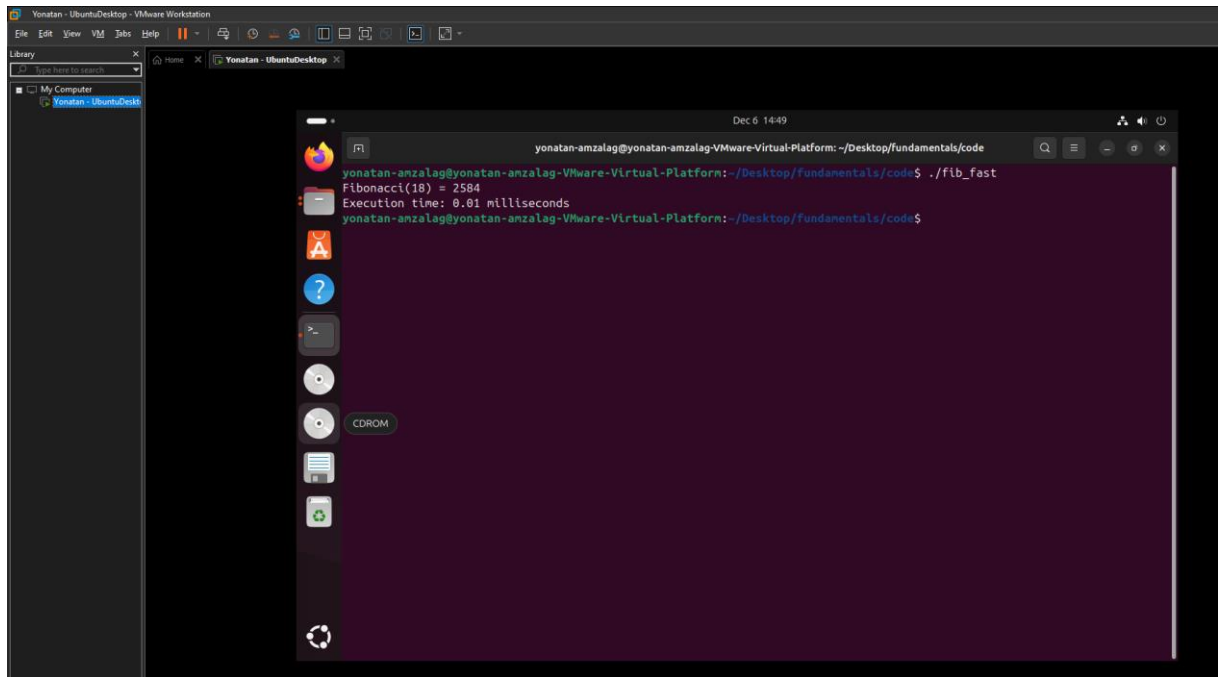
```
yonatan-amzlag@yonatan-amzlag-VMware-Virtual-Platform: ~/Desktop/fundamentals/code
yonatan-amzlag@yonatan-amzlag-VMware-Virtual-Platform: $ man gcc
yonatan-amzlag@yonatan-amzlag-VMware-Virtual-Platform: $ gcc fib -o fib_fast -O3
gcc: error: unrecognized command-line option '-O3'
yonatan-amzlag@yonatan-amzlag-VMware-Virtual-Platform: ~/Desktop/fundamentals/code$ gcc fib.c -o fib_fast -O3
gcc: error: unrecognized command-line option '-O3'
yonatan-amzlag@yonatan-amzlag-VMware-Virtual-Platform: ~/Desktop/fundamentals/code$ gcc fib.c -o fib_fast -O3
```

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

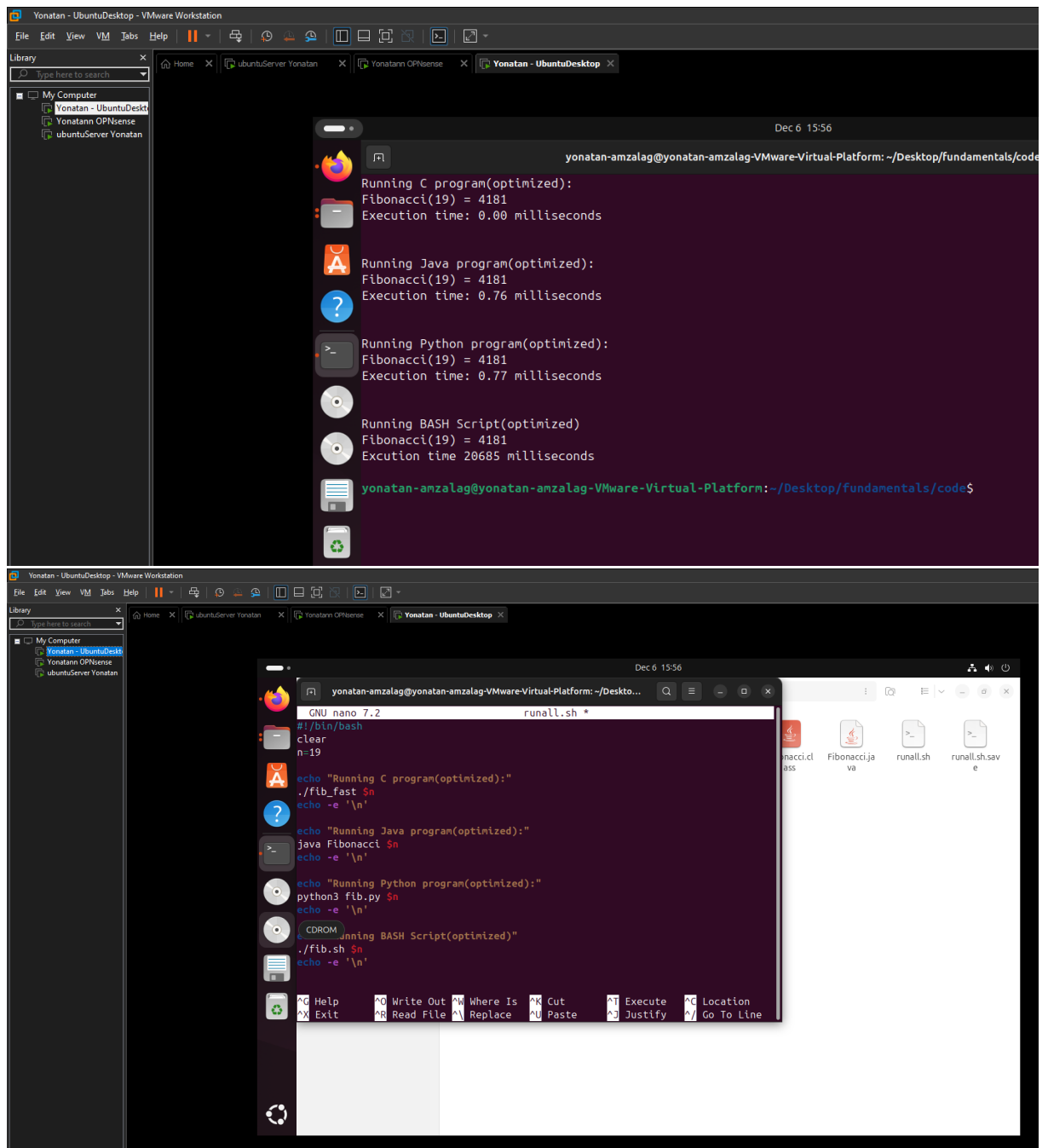


```
yonatan-amzlag@yonatan-amzlag-VMware-Virtual-Platform: ~/Desktop/fundamentals/code$ ./fib_fast
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
yonatan-amzlag@yonatan-amzlag-VMware-Virtual-Platform: ~/Desktop/fundamentals/code$
```

- 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 r1, #2
```

```
mov r2, #4
```

Loop:

End:

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

Screenshot of the completed code here.

The screenshot shows a debugger interface with two main panels. The left panel displays assembly code with the following instructions:

```
1 Main
2   mov r0, #1
3   mov r1, #2
4   mov r2, #4
5
6 Loop
7   cmp r2, #0
8   beq End
9   mul r0, r0, r1
10  sub r2, r2, #1
11  b Loop
12 End
13   b End
14 // 363634 Yonetan Amzalg
```

The right panel shows the Register Value table:

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

Below the register table is a memory dump showing hexadecimal values and their corresponding ASCII characters. The first few lines of the dump are:

```
0x00010000: 01  A0 E3 02 10 A0 E3 04 20 A0 E3 01 52 E3  B
0x00010010: 52  9A 90 01  E0 01 20 42 E2 FA FF EA  B
0x00010020: FF FF FF EA  B
0x00010030:  B
0x00010040:  B
0x00010050:  B
0x00010060:  B
0x00010070:  B
0x00010080:  B
0x00010090:  B
0x000100A0:  B
0x000100B0:  B
0x000100C0:  B
0x000100D0:  B
0x000100E0:  B
0x000100F0:  B
0x00010100:  B
0x00010110:  B
0x00010120:  B
0x00010130:  B
0x00010140:  B
0x00010150:  B
0x00010160:  B
0x00010170:  B
0x00010180:  B
0x00010190:  B
0x000101A0:  B
0x000101B0:  B
0x000101C0:  B
0x000101D0:  B
0x000101E0:  B
0x000101F0:  B
0x00010200:  B
0x00010210:  B
0x00010220:  B
0x00010230:  B
0x00010240:  B
0x00010250:  B
0x00010260:  B
```

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