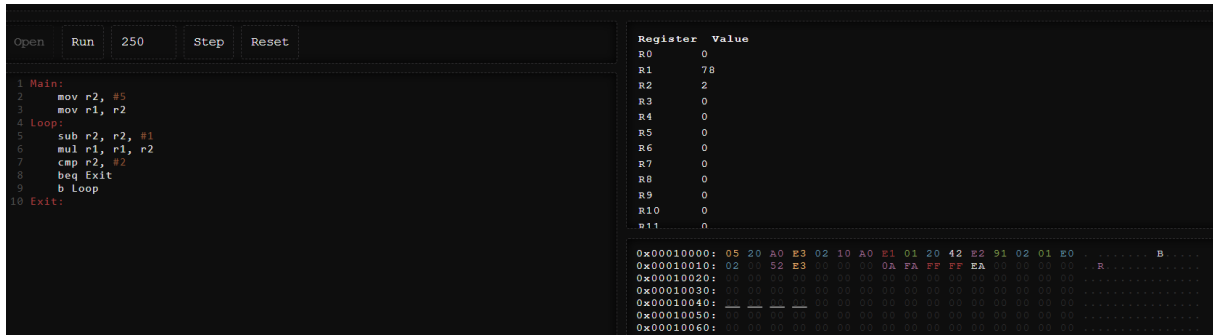


Week 4 – Software

Student number: 581124

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:

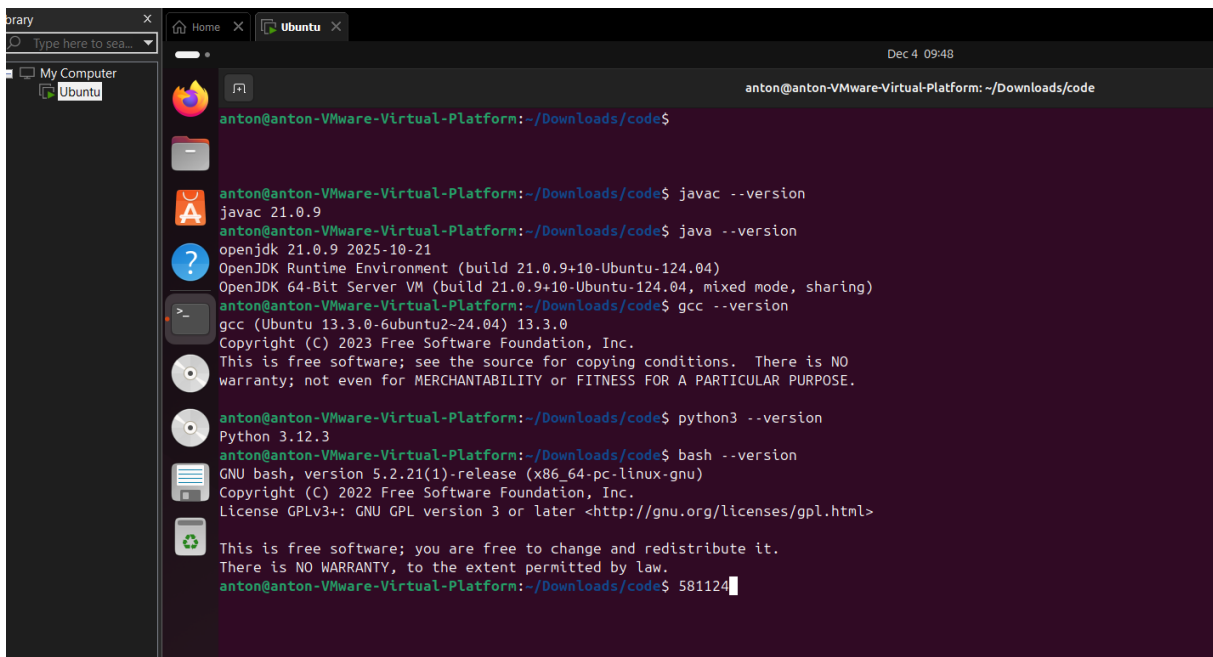
```
javac --version
```

```
java --version
```

```
gcc --version
```

```
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 must be compiled

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 is compiled into a Fibonacci.class file

Which source code files are interpreted by an interpreter?

-fib.py

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

-The fib.c, because it runs as machine code.

How do I run a Java program?

- Use this commands in bash:

1.javac Fibonacci.java

2.java Fibonacci

How do I run a Python program?

- Type in a bash: python3 fib.py

How do I run a C program?

- 1)gcc fib.c -o fib

2)./fib

How do I run a Bash script?

- ./fib.sh

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

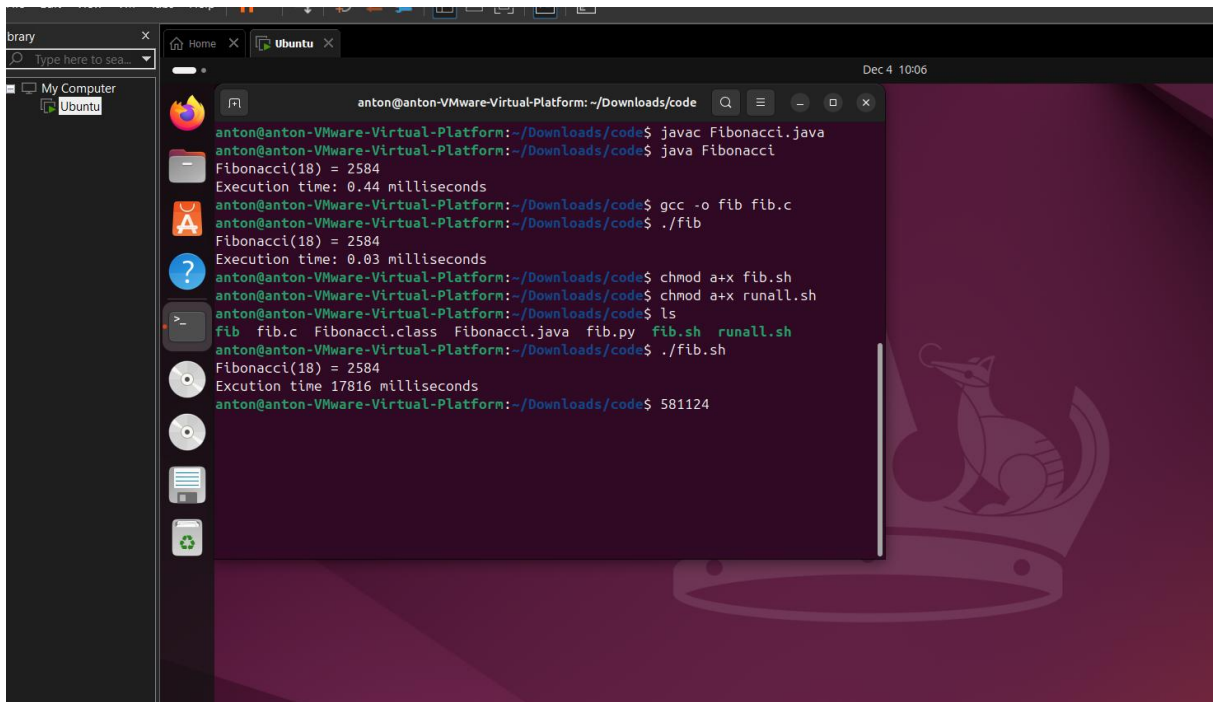
- Java will produce Fibonacci.class after compiling

- C will create executable fib or a.out

Take relevant screenshots of the following commands:

- Compile the source files where necessary
- Make them executable
- Run them

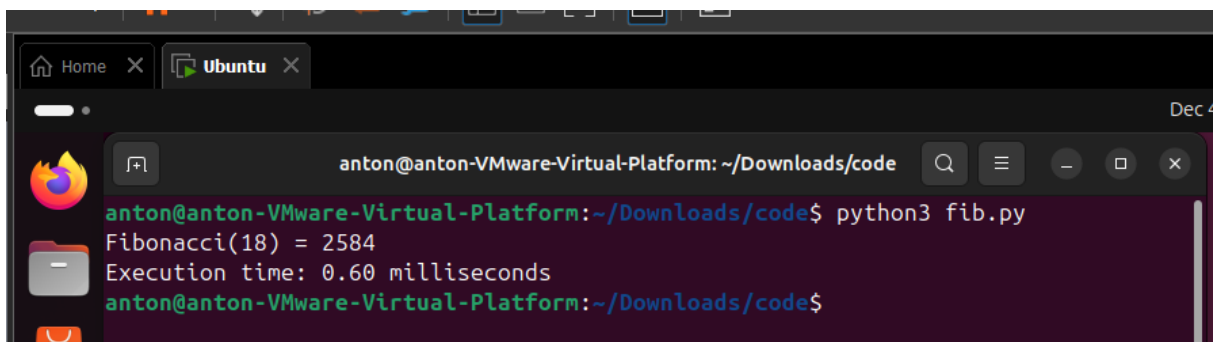
- Which (compiled) source code file performs the calculation the fastest – **C program shows the fastest performance**



```

anton@anton-VMware-Virtual-Platform: ~/Downloads/code
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.44 milliseconds
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ gcc -o fib fib.c
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ chmod a+x fib.sh
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ chmod a+x runall.sh
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 17816 milliseconds
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ 581124

```



```

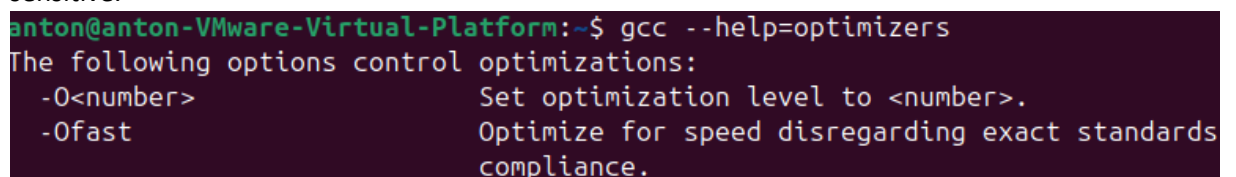
anton@anton-VMware-Virtual-Platform: ~/Downloads/code
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.60 milliseconds
anton@anton-VMware-Virtual-Platform:~/Downloads/code$

```

Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

- 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.



```

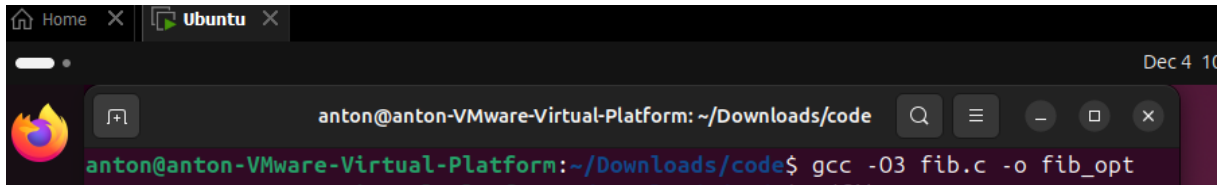
anton@anton-VMware-Virtual-Platform:~$ gcc --help=optimizers
The following options control optimizations:
-O<number>          Set optimization level to <number>.
-Ofast              Optimize for speed disregarding exact standards
                   compliance.

```

-O1 – basic optimization

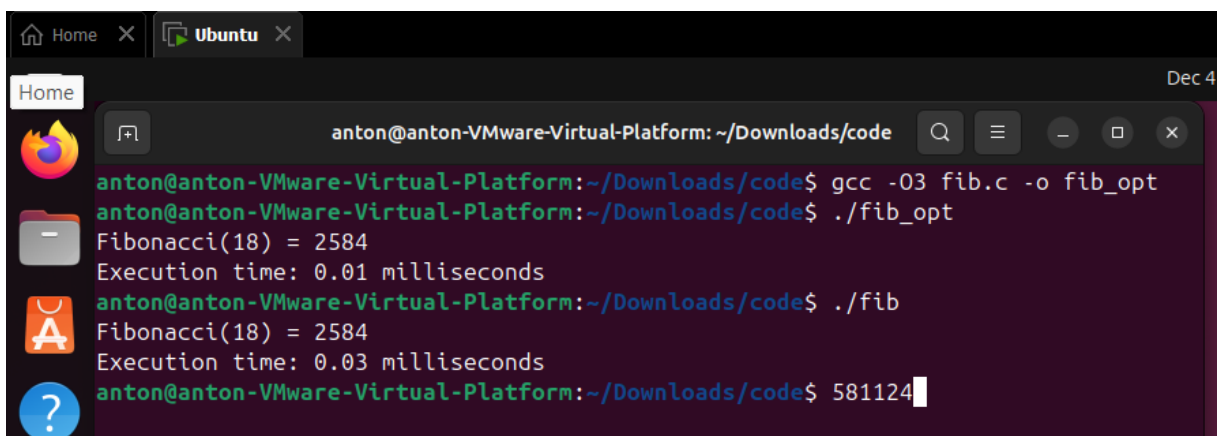
- O2 – good optimization
- O3 – fastest general optimization
- Ofast – very aggressive optimization

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



A terminal window titled 'anton@anton-VMware-Virtual-Platform: ~/Downloads/code' showing the command `gcc -O3 fib.c -o fib_opt` being executed. The terminal output is partially visible at the bottom.

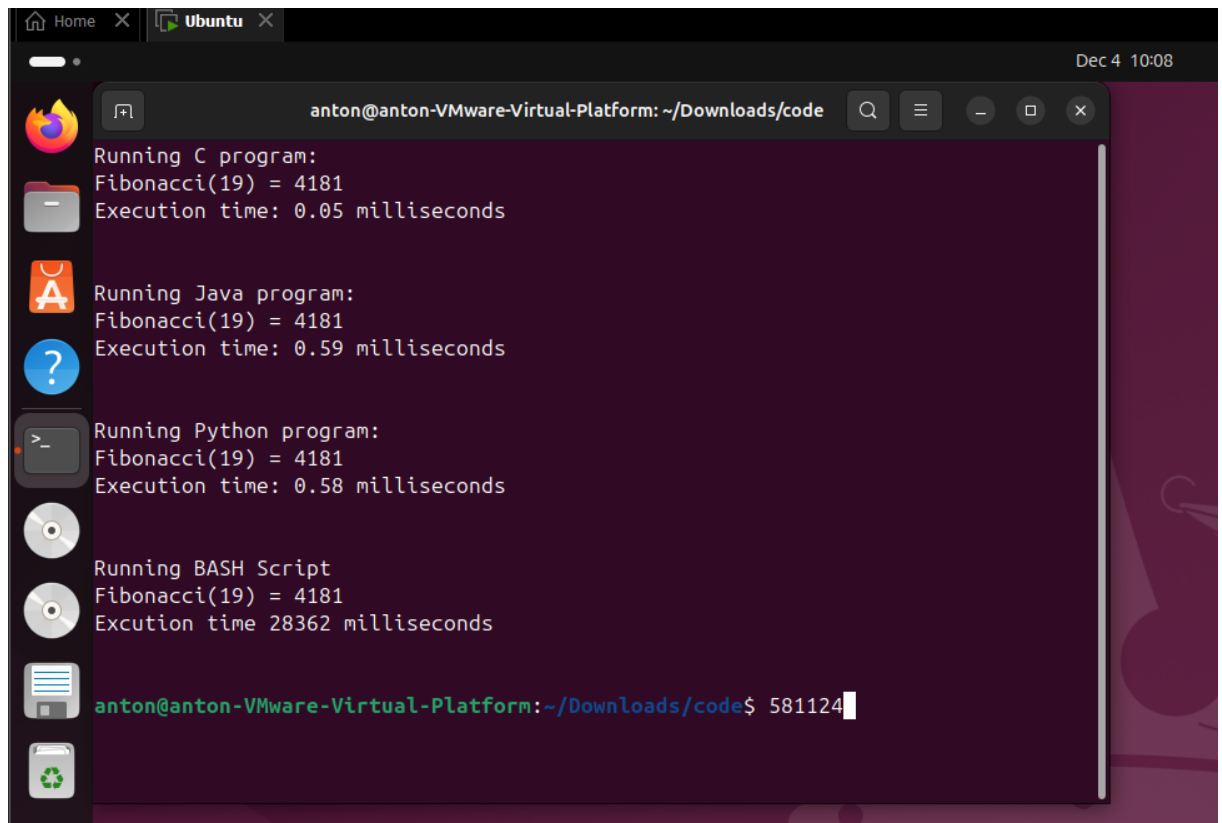
c) Run the newly compiled program. Is it true that it now performs the calculation faster?



A terminal window titled 'anton@anton-VMware-Virtual-Platform: ~/Downloads/code' showing the execution of the compiled program. The output shows the Fibonacci(18) result and execution time for both the original and compiled versions.

```
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 fib.c -o fib_opt
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
anton@anton-VMware-Virtual-Platform:~/Downloads/code$ 581124
```

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.



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.

OakSim

Open

Run

250

Step

Reset

```
1 Main:
2   mov r0, #1
3   mov r1, #2
4   mov r2, #4
5 Loop:
6   mul r0, r0, r1
7   subs r2, r2, #1
8   beq Exit
9   b Loop
10 Exit:
```

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

0x00010000: 01 00 A0

0x00010010: 01 20 54

0x00010020: 00 00 00

0x00010030: 00 00 00