

Week 4 – Software

Student number: 585732

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim interface with the assembly code for calculating a factorial. The code initializes R2 with 5, R1 with 1, and then enters a loop where it multiplies R1 by R2, decrements R2, and loops back until R2 is 1. The memory dump shows the state of memory from address 0x000010000 to 0x000010210.

```
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:
```

Register	Value
R0	0
R1	78
R2	1
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0

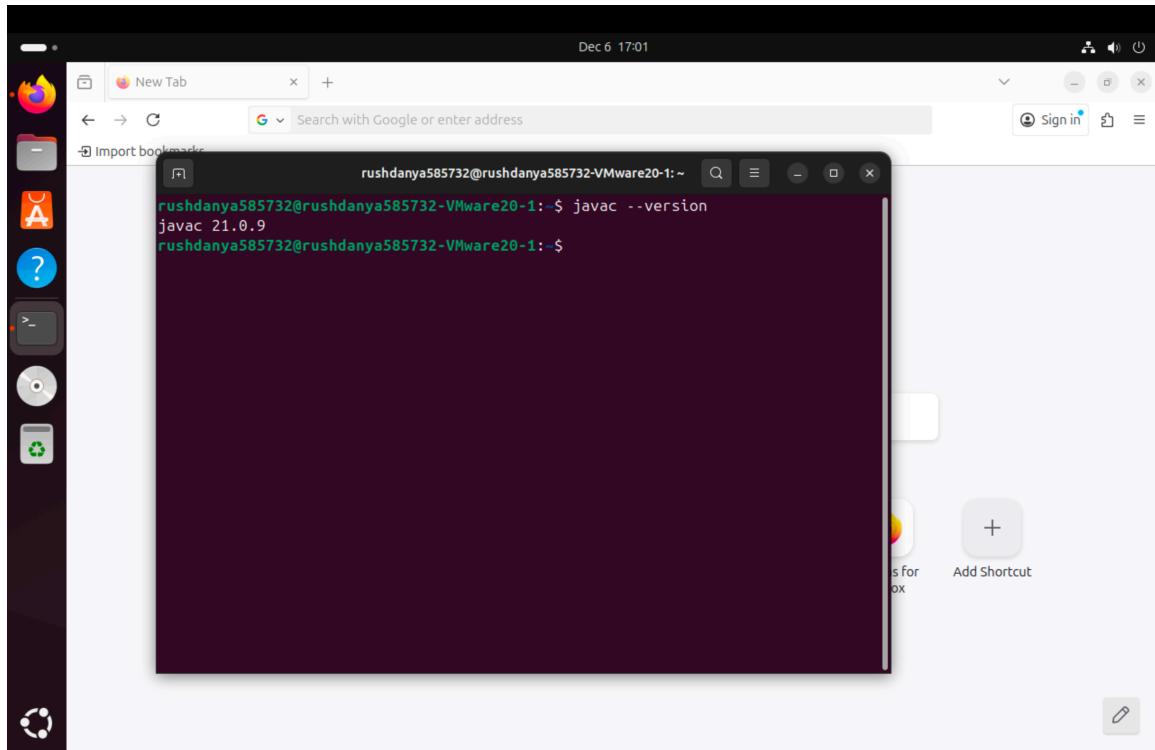
Memory Dump (addresses 0x000010000 - 0x000010210):

Address	Value
0x000010000	05 20 A0 E3 01 10 A0 E3 01 00 52 E3 02 00 00 0A . . . B R
0x000010010	91 02 01 E0 01 20 42 E2 FA FF FF EA 00 00 00 00 00 . . . B
0x000010020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000100A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000100B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000100C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000100D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000100E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000100F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010170	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000101A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000101B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000101C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000101D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000101E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x0000101F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010200	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B
0x000010210	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . B

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

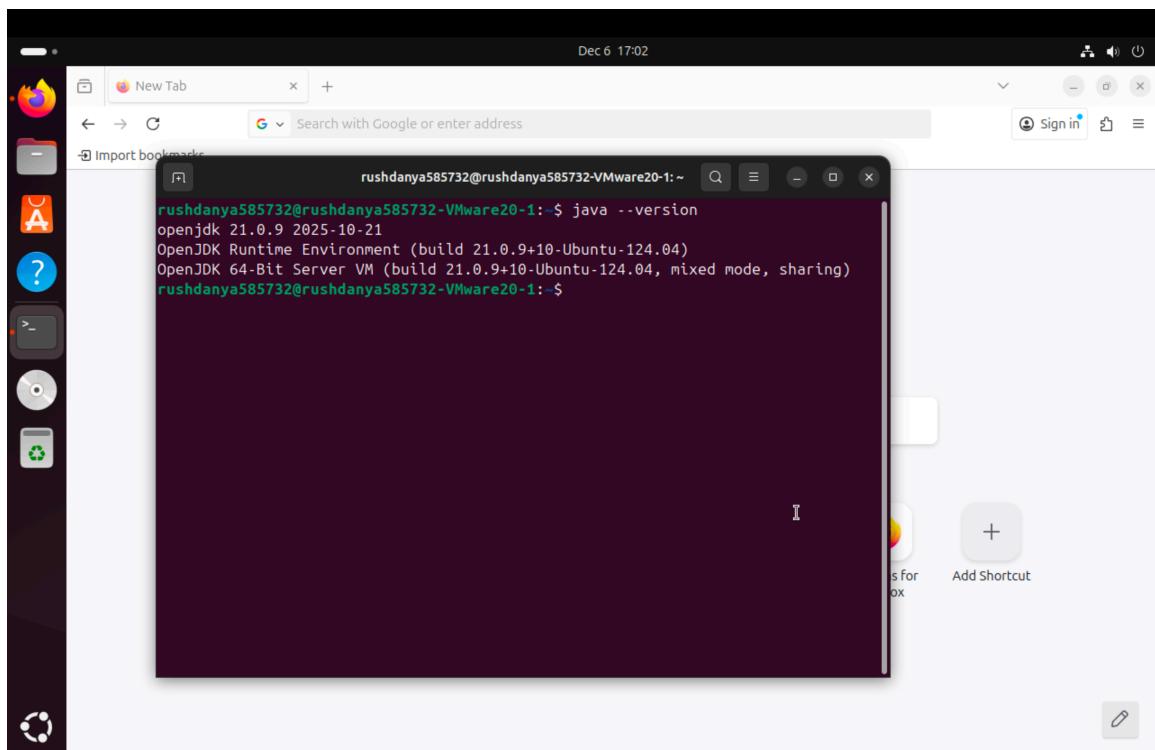
javac –version



A screenshot of a Linux desktop environment showing a terminal window. The terminal window title is "rushdanya585732@rushdanya585732-VMware20-1:~". The terminal content shows the command "javac --version" being run, which outputs "javac 21.0.9" and then exits. The desktop background is dark, and there are icons for a file manager, terminal, and browser in the dock.

```
rushdanya585732@rushdanya585732-VMware20-1:~$ javac --version
javac 21.0.9
rushdanya585732@rushdanya585732-VMware20-1:~$
```

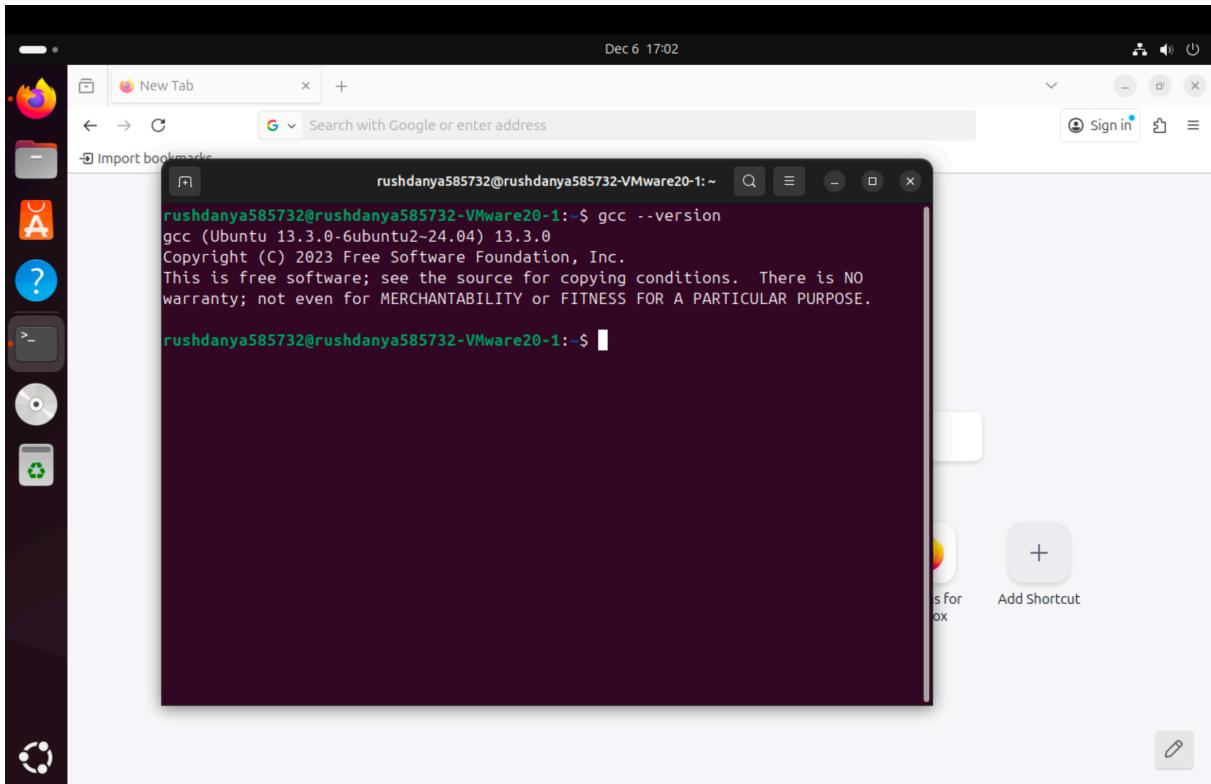
java –version



A screenshot of a Linux desktop environment showing a terminal window. The terminal window title is "rushdanya585732@rushdanya585732-VMware20-1:~". The terminal content shows the command "java --version" being run, which outputs information about OpenJDK Runtime Environment and OpenJDK 64-Bit Server VM, both build 21.0.9+10-Ubuntu-124.04. The desktop background is dark, and there are icons for a file manager, terminal, and browser in the dock.

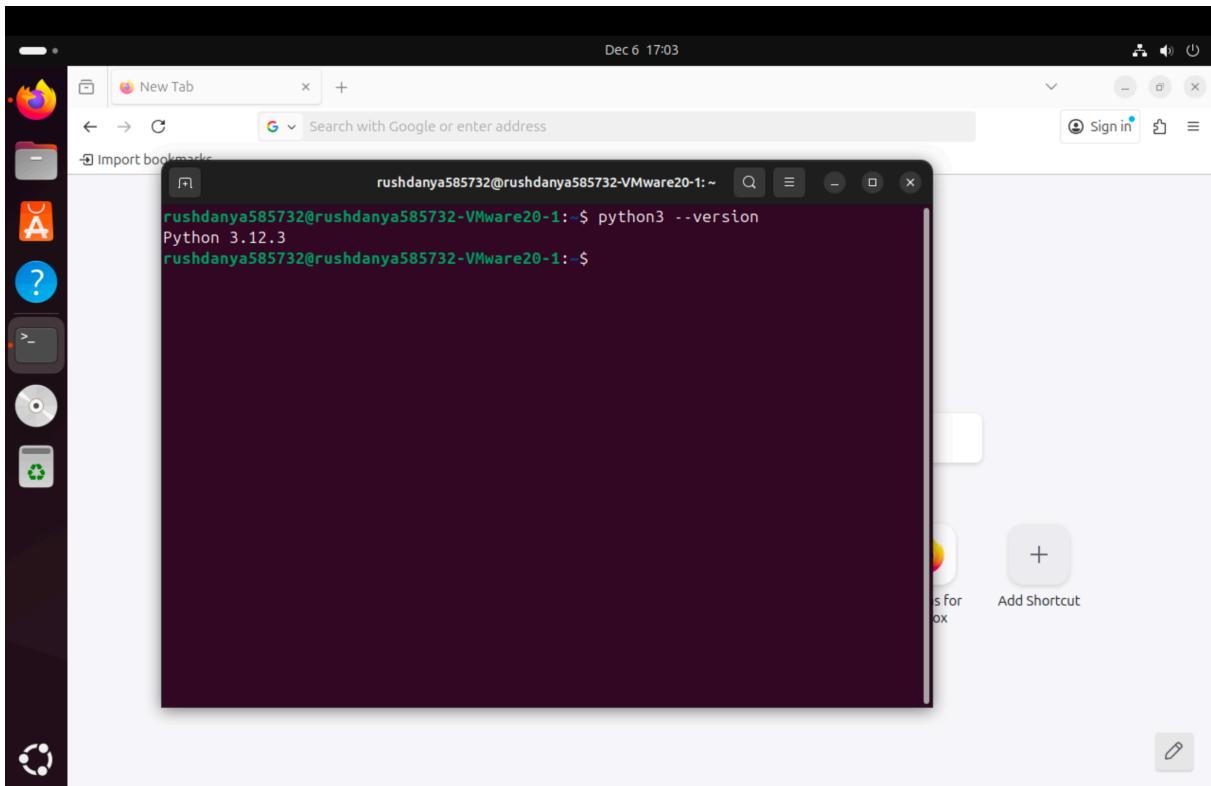
```
rushdanya585732@rushdanya585732-VMware20-1:~$ java --version
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
rushdanya585732@rushdanya585732-VMware20-1:~$
```

gcc --version



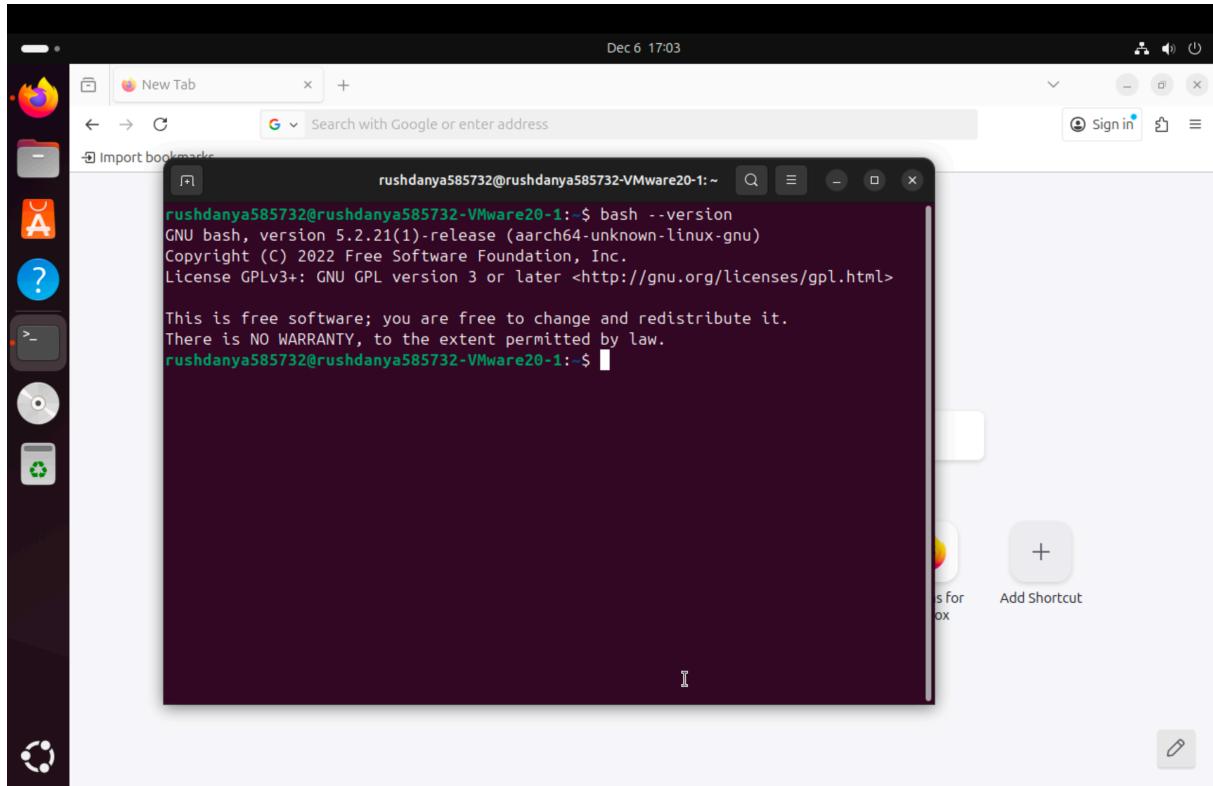
A screenshot of a Linux desktop environment. In the center is a terminal window titled "rushdanya585732@rushdanya585732-VMware20-1:~". The terminal displays the command "gcc --version" followed by its output: "gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0 Copyright (C) 2023 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE." The desktop background is dark, and the taskbar at the bottom has icons for various applications like a browser, file manager, and system tools.

python3 --version



A screenshot of a Linux desktop environment, identical to the one above, showing a terminal window with the command "python3 --version". The output shows Python 3.12.3. The desktop interface, including the taskbar with application icons, is visible.

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 C program (fib.c, after compilation)

Which source code files are compiled to byte code?

= The Java program (Fibonacci.java into Fibonacci.class)

Which source code files are interpreted by an interpreter?

= The Python program(fib.py) and the Bash script (fib.sh)

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

= The compiled C program.

How do I run a Java program?

= 1. Compile: javac Fibonacci.java

2. Run: java Fibonacci

How do I run a Python program?

= python3 fib.py

How do I run a C program?

= Compile: gcc fib.c -o fib_c

How do I run a Bash script?

= 1. Make executable: chmod a+x fib.sh

2. Run: ./fib.sh (or bash fib.sh)

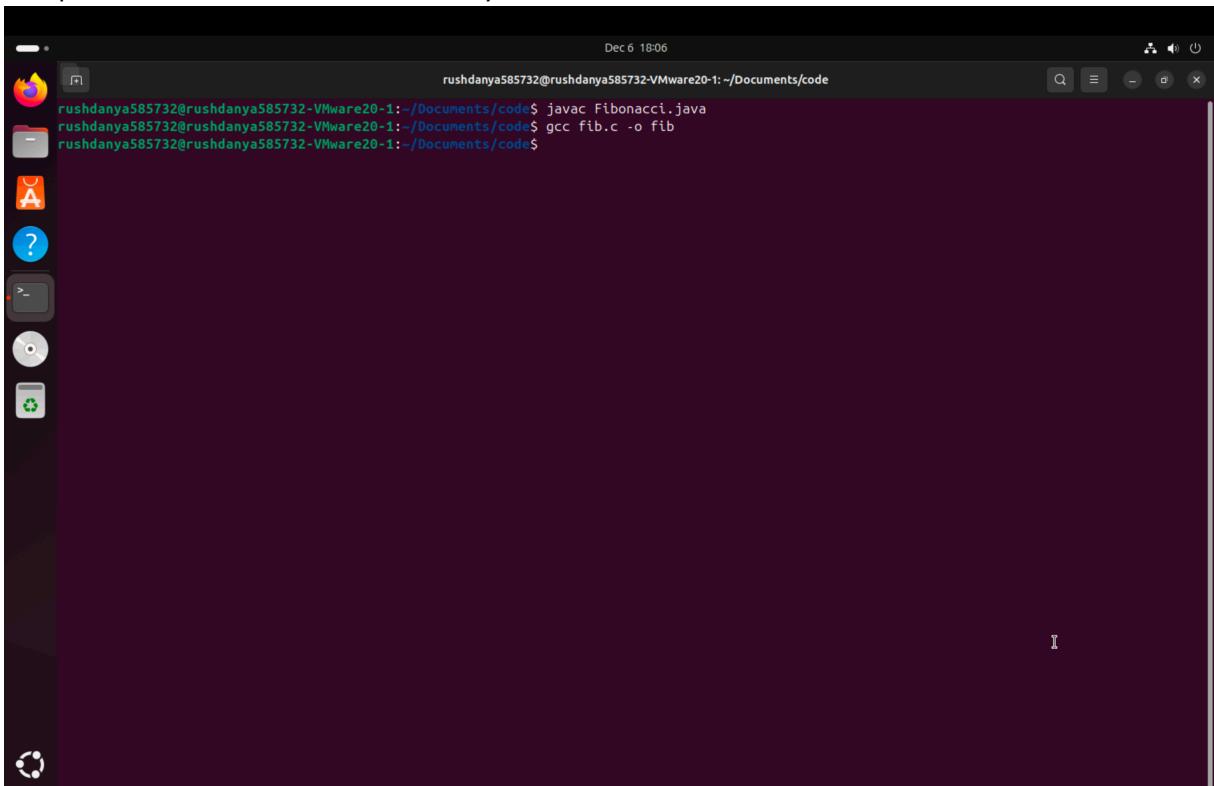
If I compile the above source code, will a new file be created? If so, which file?

= Compiling Fibonacci.java creates Fibonacci.class

Compiling fib.c (example: gcc fib.c -o fib_c) creates the executable fib_c .

Take relevant screenshots of the following commands:

- Compile the source files where necessary

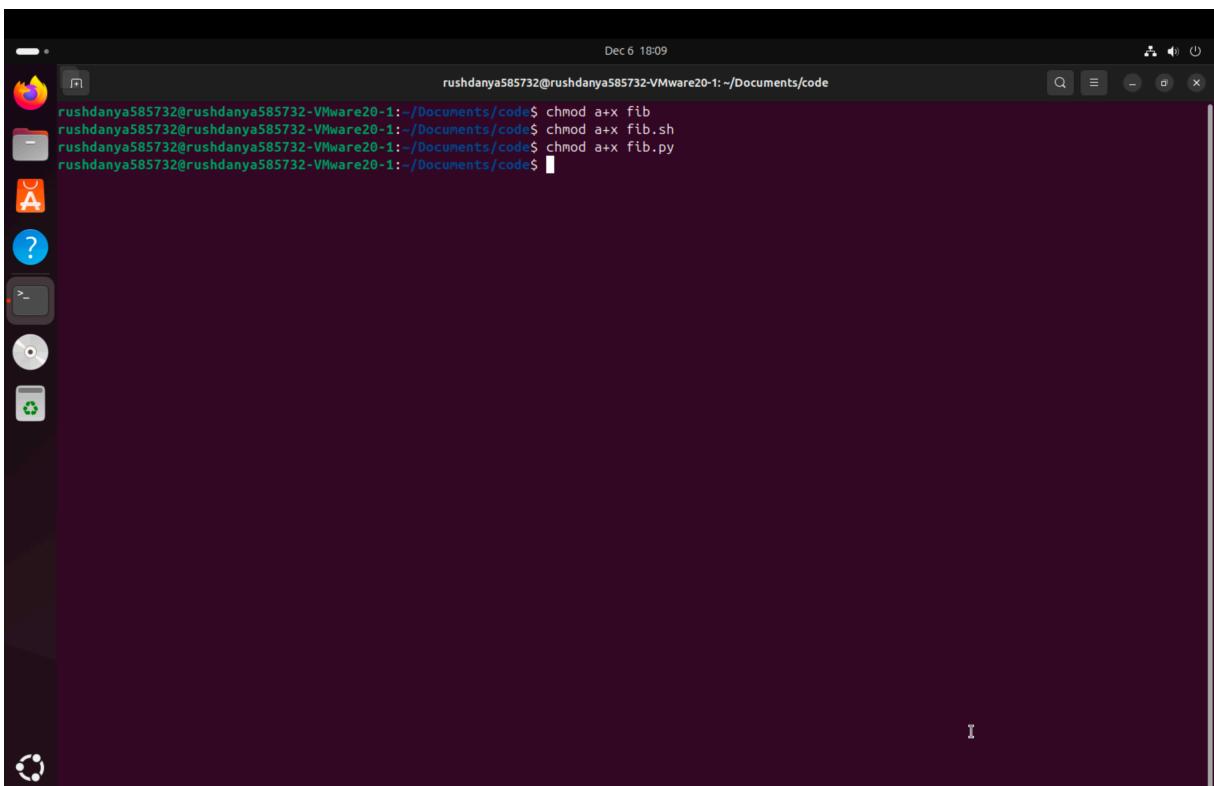


A screenshot of a Linux desktop environment showing a terminal window. The terminal window has a dark background and contains the following command history:

```
Dec 6 18:06
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ javac Fibonacci.java
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ gcc fib.c -o fib
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$
```

The terminal window is part of a desktop interface with icons for file, application, and system functions visible on the left.

- Make them executable

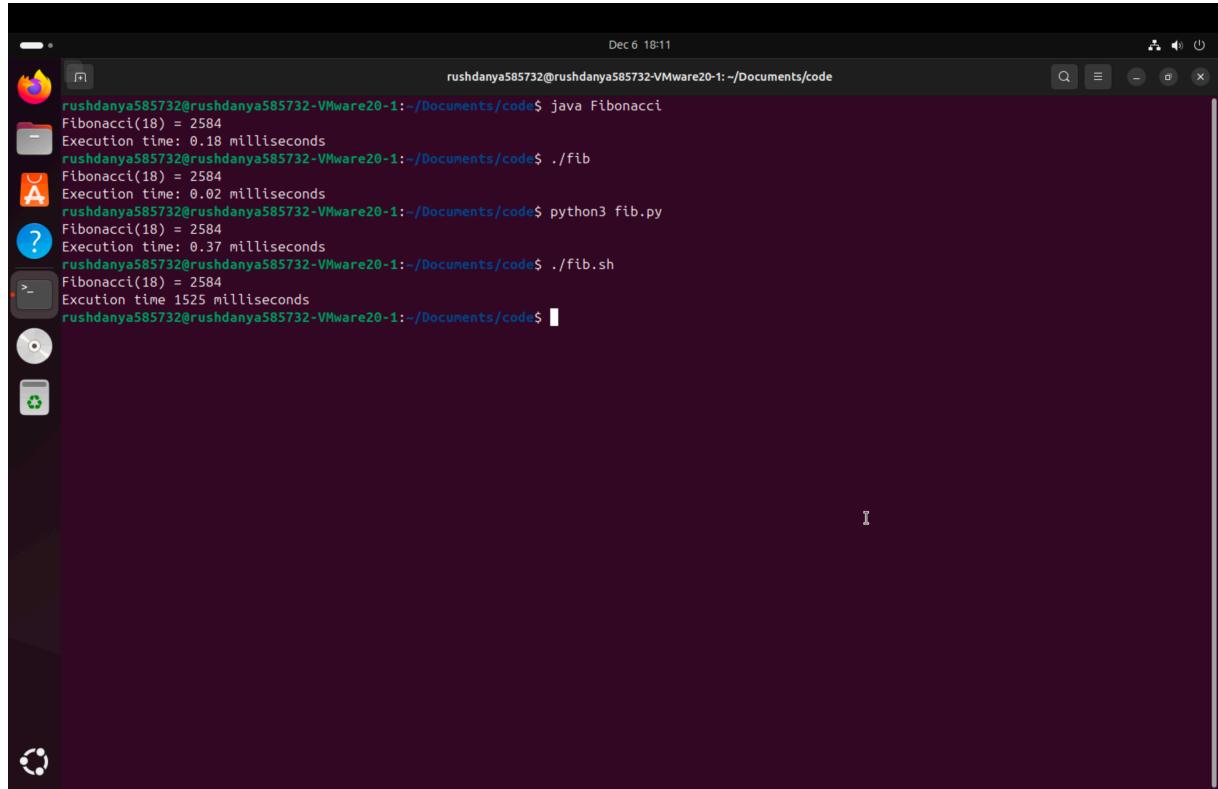


A screenshot of a Linux desktop environment showing a terminal window. The terminal window has a dark background and contains the following command history:

```
Dec 6 18:09
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ chmod a+x fib
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ chmod a+x fib.sh
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ chmod a+x fib.py
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$
```

The terminal window is part of a desktop interface with icons for file, application, and system functions visible on the left.

- Run them



```

Dec 6 18:11
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.18 milliseconds
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.37 milliseconds
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 1525 milliseconds
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ 
```

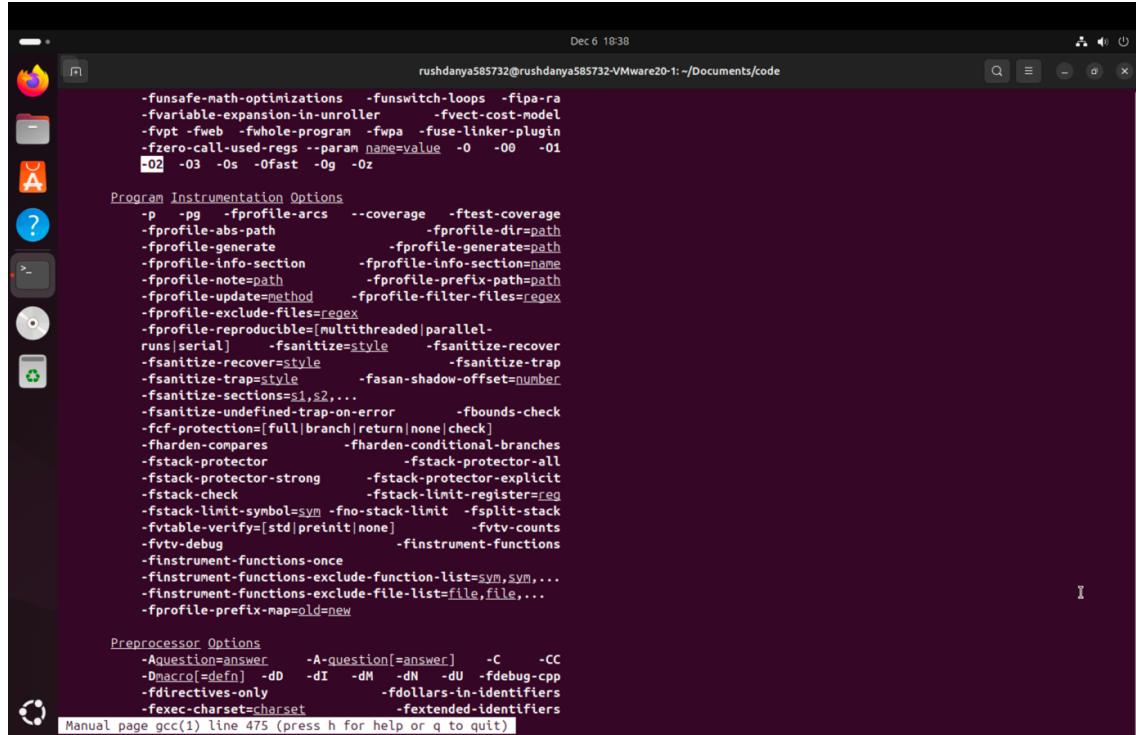
- Which (compiled) source code file performs the calculation the fastest?

After compiling and running all the programs, we can see that the C version turned out to be the fastest. In my test, it completed the Fibonacci calculation in about 0.02 milliseconds, while the Java version took about 0.18 milliseconds. So the compiled souce code file that performs the calculation the fastest is fib.c.

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.



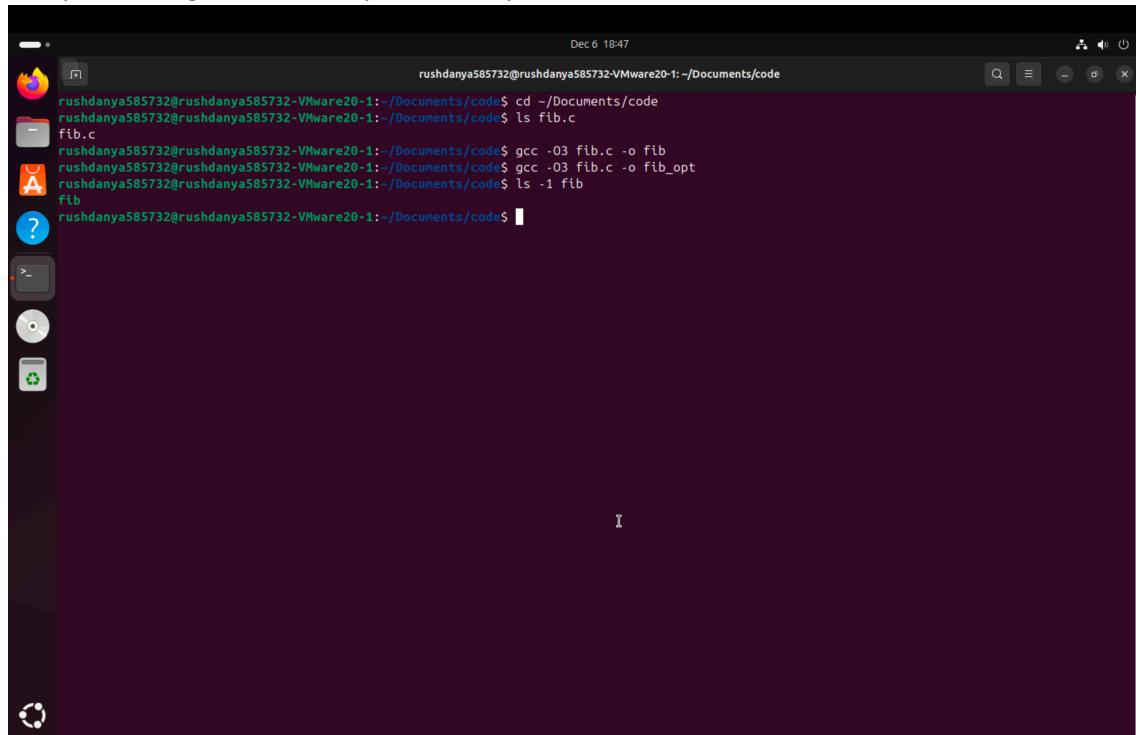
The screenshot shows a terminal window with the following content:

```
Dec 6 18:38
rushdanya585732@rushdanya585732:~/Documents/code$ man gcc
... (many lines of text from the man page) ...
Program Instrumentation Options
  -P -pg   -fprofile-arcs  --coverage  -ftest-coverage
  -fprofile-abs-path          -fprofile-dir=path
  -fprofile-generate         -fprofile-generate=path
  -fprofile-info-section     -fprofile-info-section=name
  -fprofile-note=path        -fprofile-prefix-path=path
  -fprofile-update=method    -fprofile-filter-files=regex
  -fprofile-exclude-files=regex
  -fprofile-reproducible=[multithreaded|parallel-
    runs|serial]  -fsanitize=style  -fsanitize-recover
  -fsanitize-recover=style   -fsanitize-trap
  -fsanitize-trap=style     -fsan-shadow-offset=number
  -fsanitize-sections=s1,s2,...
  -fsanitize-undefined-trap-on-error  -fbounds-check
  -fcf-protection=[full|branch|return|none|check]
  -fharden-compare           -fharden-conditional-branches
  -fstack-protector          -fstack-protector-all
  -fstack-protector-strong   -fstack-protector-explicit
  -fstack-check               -fstack-limit-register=reg
  -fstack-limit-symbol=sym  -fno-stack-limit  -fsplit-stack
  -fvtable-verify=[std|preint|none]  -fvtx-counts
  -fvtx-debug                -finstrument-functions
  -finstrument-functions-once
  -finstrument-functions-exclude-function-list=sym,sym,...
  -finstrument-functions-exclude-file-list=file,file,...
  -fprofile-prefix-map=old=new

Preprocessor Options
  -Aquestion=answer  -Aquestion[=answer]  -C  -CC
  -Dmacro[=defn]  -D0  -DI  -DM  -DN  -DU  -fdebug-cpp
  -f directives-only          -fdollars-in-identifiers
  -fexec-charset=charset    -fextended-identifiers
  -fmanual

Manual page gcc(1) line 475 (press h for help or q to quit)
```

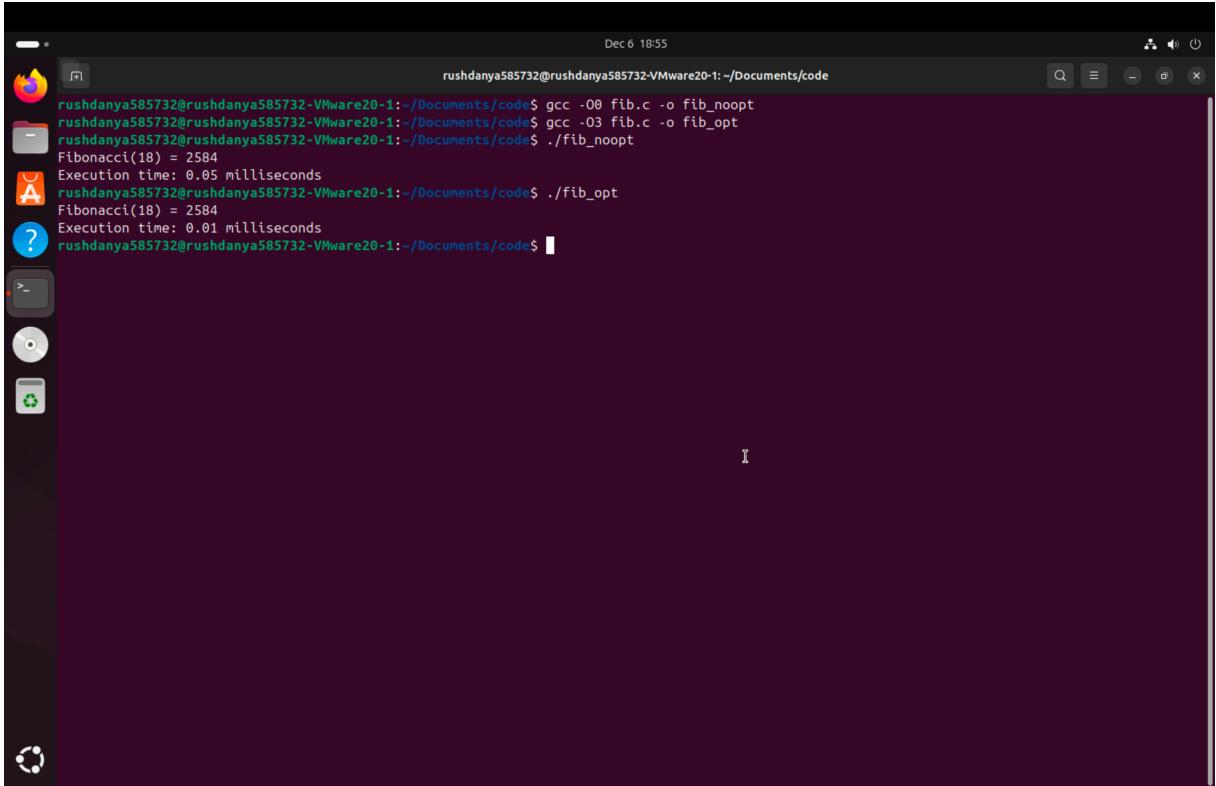
- Compile **fib.c** again with the optimization parameters



The screenshot shows a terminal window with the following content:

```
Dec 6 18:47
rushdanya585732@rushdanya585732:~/Documents/code$ cd ~/Documents/code
rushdanya585732@rushdanya585732:~/Documents/code$ ls fib.c
fib.c
rushdanya585732@rushdanya585732:~/Documents/code$ gcc -O3 fib.c -o fib
rushdanya585732@rushdanya585732:~/Documents/code$ gcc -O3 fib.c -o fib_opt
fib_opt
rushdanya585732@rushdanya585732:~/Documents/code$ ls -l fib
fib
rushdanya585732@rushdanya585732:~/Documents/code$
```

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



The screenshot shows a terminal window on a dark-themed desktop environment. The terminal title bar reads "rushdanya585732@rushdanya585732-VMware20-1: ~/Documents/code". The terminal window displays the following command-line session:

```
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ gcc -O0 fib.c -o fib_noopt
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ gcc -O3 fib.c -o fib_opt
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ ./fib_noopt
Fibonacci(18) = 2584
Execution time: 0.05 milliseconds
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$
```

Yes, it performs the calculation faster after compiling with optimisations. In my run, the unoptimised version (fib_noopt) took about 0.05 ms, while the optimised version (fib_opt, compiled with -O3) took about 0.01 ms for the same input (Fibonacci(18) = 2584). This shows the optimised compiled program 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.

```

Dec 6 19:12
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ cd ~/Documents/code
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ ls
fib fib_noopt Fibonacci.java fib.py -O1
fib.c Fibonacci.class fib_opt fib.sh runall.sh
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ nano runall.sh
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ chmod +x runall.sh
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ ./runall.sh
*** C version (fib_opt) ===
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

*** Java version (Fibonacci) ===
Fibonacci(18) = 2584
Execution time: 0.19 milliseconds

*** Python version (fib.py) ===
Fibonacci(18) = 2584
Execution time: 0.19 milliseconds

*** Bash version (fib.sh) ===
Fibonacci(18) = 2584
Execution time 1540 milliseconds

rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ 

```

```

Dec 6 19:14
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ cat runall.sh
#!/bin/bash
echo "==== C version (fib_opt) ===="
./fib_opt
echo
echo "==== Java version (Fibonacci) ===="
java Fibonacci
echo
echo "==== Python version (fib.py) ===="
python3 fib.py
echo
echo "==== Bash version (fib.sh) ===="
./fib.sh
echo
rushdanya585732@rushdanya585732-VMware20-1:~/Documents/code$ ./runall.sh
*** C version (fib_opt) ===
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

*** Java version (Fibonacci) ===
Fibonacci(18) = 2584
Execution time: 0.18 milliseconds

*** Python version (fib.py) ===
Fibonacci(18) = 2584
Execution time: 0.18 milliseconds

*** Bash version (fib.sh) ===
Fibonacci(18) = 2584
Execution time 1537 milliseconds

rushdanya585732@rushdanya585732-VMware20-1:~/Documents/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  
mov r0, #1
```

Loop:

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

End:

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

Screenshot of the completed code here.

The screenshot shows the OakSim debugger interface. On the left, the assembly code is displayed:

```
1 Main:  
2     mov r1, #2  
3     mov r2, #4  
4     mov r0, #1  
5  
6 Loop:  
7     mul r0, r0, r1  
8     subs r2, r2, #1  
9     bne Loop  
10  
11 End:
```

On the right, the register values are listed:

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

The memory dump area shows the state of memory starting at address 0x000010000. The first few bytes are 02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 90 01 00 E0, followed by several zeros. The bottom of the screen displays the copyright notice "© 2017".