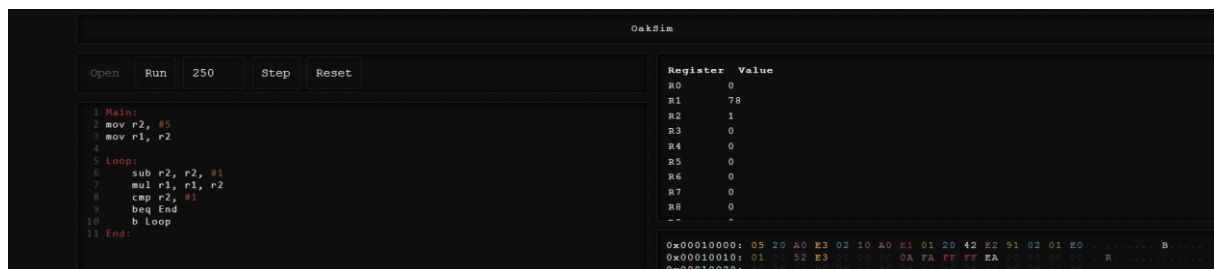


Template Week 4 – Software

Student number: 564595

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

```
reza-hekmatirad@reza-hekmatirad-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
reza-hekmatirad@reza-hekmatirad-VMware-Virtual-Platform:~$ python3 --version
Python 3.11.6
reza-hekmatirad@reza-hekmatirad-VMware-Virtual-Platform:~$ bash --version
GNU bash, version 5.2.21(1)-release (x86_64-pc-linux-gnu)
Copyright (c) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
reza-hekmatirad@reza-hekmatirad-VMware-Virtual-Platform:~$ 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.
reza-hekmatirad@reza-hekmatirad-VMware-Virtual-Platform:~$ 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)
reza-hekmatirad@reza-hekmatirad-VMware-Virtual-Platform:~$
```

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?

- fib.c

Which source code files are compiled to byte code?

-Fibonacci.java

Which source code files are interpreted by an interpreter?

- fib.py and fib.py

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?

-First compile it via javac (file name.java) then run it via java(file name)

How do I run a Python program?

-It does not need to be compiled since Python is interpreted so just do the python3 (file name.py)

How do I run a C program?

-First compile it via gcc -o (file name)(file name.c) then run it via ./(file name)

How do I run a Bash script?

-It does not need to be compiled since it is interpreted so first make it executable(chmod + x (file name)) and then just do the bash(file name.sh)

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

For .c files → Yes → fib.c turns into fib

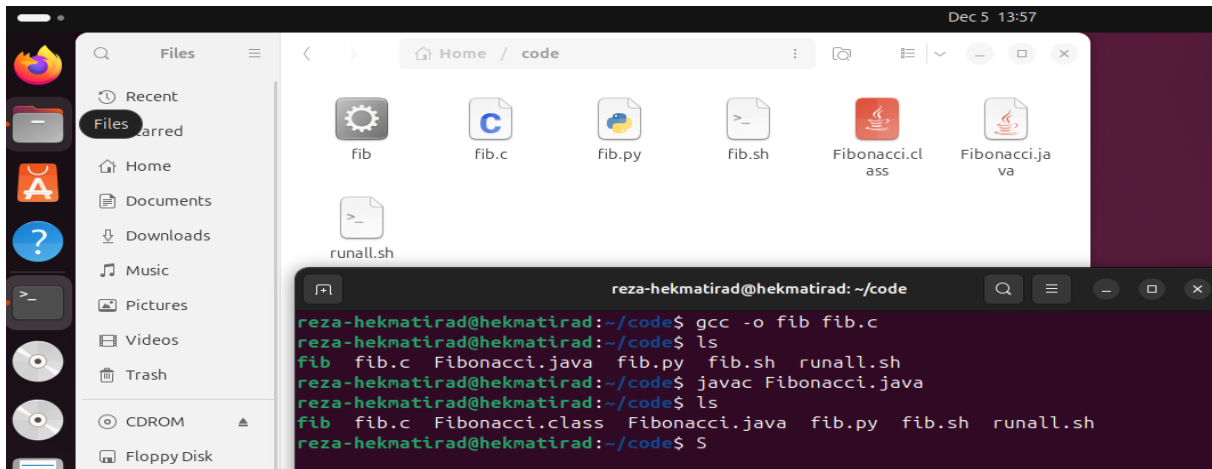
For .java → Yes → Fibonacci.java turns into Fibonacci.class

For .py → No

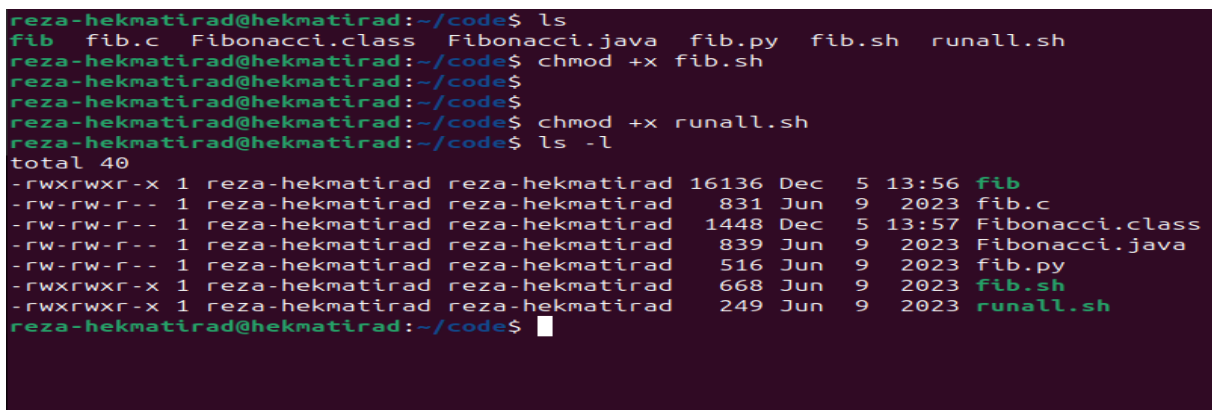
For .sh or .bash → No

Take relevant screenshots of the following commands:

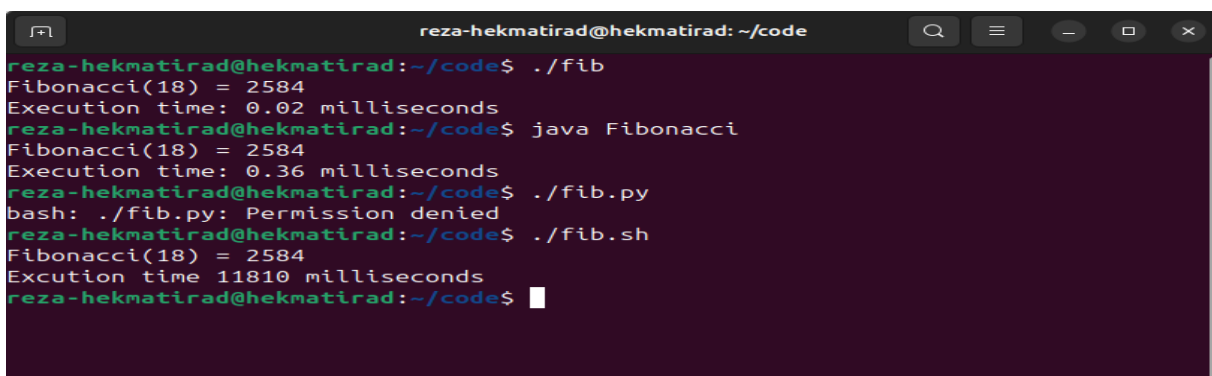
- Compile the source files where necessary:



- Make them executable



- Run them



```
reza-hekmatirad@hekmatirad: ~/code
Running C program:
Fibonacci(19) = 4181
Execution time: 0.03 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.52 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 1.02 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 18798 milliseconds

reza-hekmatirad@hekmatirad:~/code$
```

- Which (compiled) source code file performs the calculation the fastest?
- Obviously `.c` source code

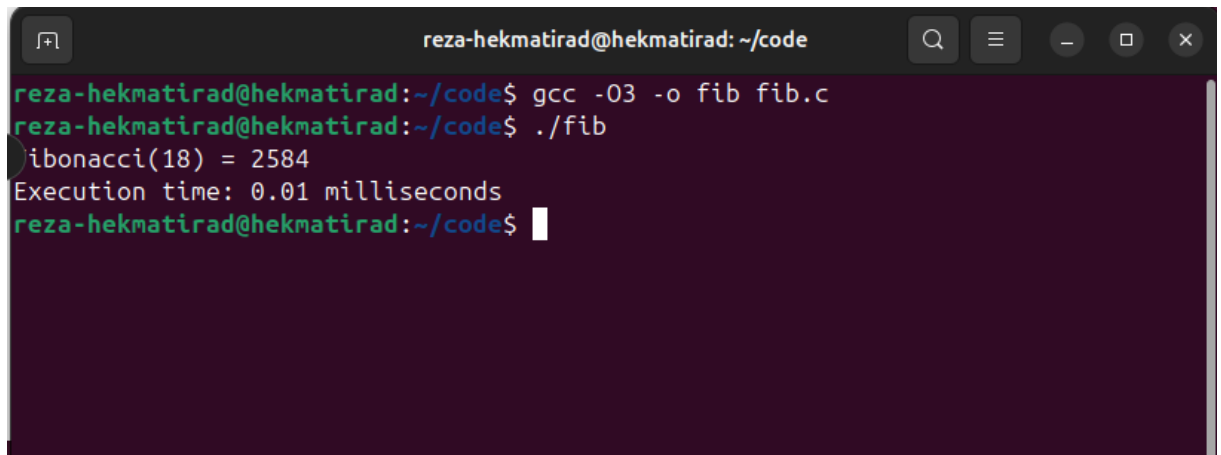
Assignment 4.4: Optimize

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.
-gcc -O3 -o fib fib.c
- Compile **fib.c** again with the optimization parameters

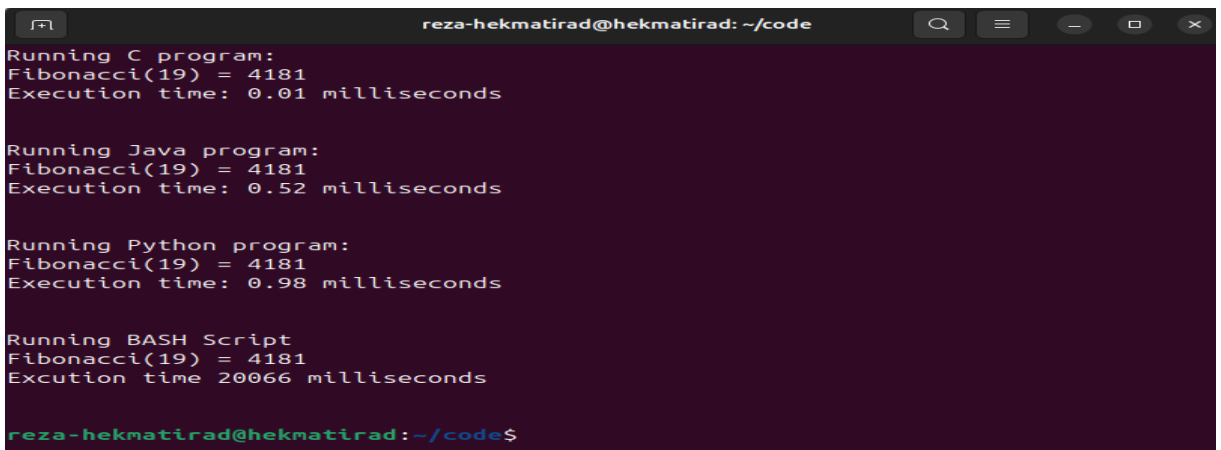
```
reza-hekmatirad@hekmatirad: ~/code
reza-hekmatirad@hekmatirad:~/code$ gcc -O3 -o fib fib.c
reza-hekmatirad@hekmatirad:~/code$ S
```

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



```
reza-hekmatirad@hekmatirad: ~/code
reza-hekmatirad@hekmatirad:~/code$ gcc -O3 -o fib fib.c
reza-hekmatirad@hekmatirad:~/code$ ./fib
fibonacci(18) = 2584
Execution time: 0.01 milliseconds
reza-hekmatirad@hekmatirad:~/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.



```
reza-hekmatirad@hekmatirad: ~/code
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.52 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.98 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 20066 milliseconds

reza-hekmatirad@hekmatirad:~/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
```

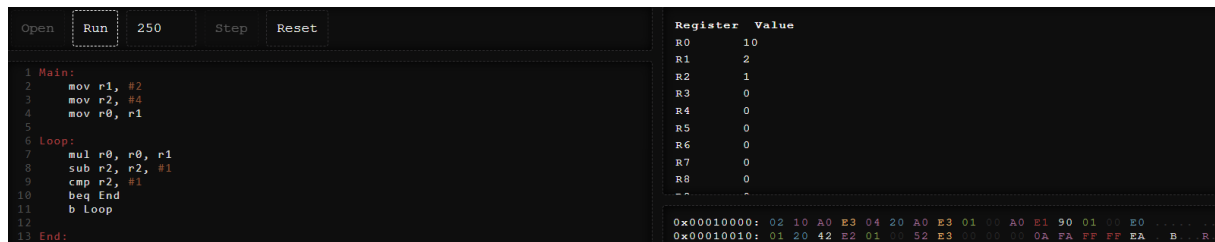
```
mov r2, #4
```

Loop:

End:

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

Screenshot of the completed code here.



The screenshot shows an assembly code editor with a dark background. At the top, there are buttons for 'Open', 'Run', '250', 'Step', and 'Reset'. The 'Run' button is highlighted. Below the buttons, the assembly code is displayed in a monospaced font. The code is as follows:

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

To the right of the code, there is a 'Register Value' window. It lists registers R0 through R8 and their current values. R0 is 10, R1 is 2, R2 is 1, and R3 through R8 are 0. Below the register window, there is a memory dump showing two lines of memory addresses and their corresponding values in hexadecimal and ASCII.

| Register | Value |
|----------|-------|
| R0 | 10 |
| R1 | 2 |
| R2 | 1 |
| R3 | 0 |
| R4 | 0 |
| R5 | 0 |
| R6 | 0 |
| R7 | 0 |
| R8 | 0 |

Memory dump:

```
0x00010000: 02 10 A0 E3 04 20 A0 E3 01 80 A0 E1 90 01 00 E0 ...
0x00010010: 01 20 42 E2 01 00 52 E3 00 00 00 0A FA FF FF EA B R
```

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