

Week 4 – Software

Student number: 588406

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim interface running on a Mac OS X desktop. The window title is "OakSim". The assembly code in the main pane is as follows:

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

To the right of the assembly code is a register dump table:

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

Below the assembly code, there is a stack trace:

```
abort() at
jsStackTrace@https://wunkolo.github.io/Oak
arm.min.js:5:18821
stackTrace@https://wunkolo.github.io/OakSi
arm.min.js:5:19004
abort@https://wunkolo.github.io/OakSim/lib
```

The bottom of the window displays the copyright notice "© 2017".

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac –version

A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window. The terminal window title is "Kaja_588406Ubuntu 64-bit Arm 24.04.3". The terminal shows the command "java --version" being run, which outputs "javac 21.0.9" and "javac 21.0.9".

```
Dec 9 17:15
kej588406@kej588406-Vmware20:~$ javac --version
javac 21.0.9
kej588406@kej588406-Vmware20:~$
```

`java --version`

A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window. The terminal window title is "Kaja_588406Ubuntu 64-bit Arm 24.04.3". The terminal shows the command "java --version" being run, which outputs "javac 21.0.9" and "javac 21.0.9". It also shows the command "java -version" being run, which outputs "openjdk 21.0.9 2025-10-21" and "OpenJDK Runtime Environment (build 21.0.9+18-Ubuntu-124.04)".

```
Dec 9 17:15
kej588406@kej588406-Vmware20:~$ javac --version
javac 21.0.9
kej588406@kej588406-Vmware20:~$ java --version
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+18-Ubuntu-124.04)
kej588406@kej588406-Vmware20:~$
```

`gcc --version`

A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window. The terminal window title is "Kaja_588406Ubuntu 64-bit Arm 24.04.3". The terminal shows the command "java --version" being run, which outputs "javac 21.0.9" and "javac 21.0.9". It also shows the command "java -version" being run, which outputs "openjdk 21.0.9 2025-10-21" and "OpenJDK Runtime Environment (build 21.0.9+18-Ubuntu-124.04)". Finally, it shows the command "gcc --version" being run, which outputs "gcc (Ubuntu 13.3.0-1ubuntu1.1) 13.3.0" and "Copyright (C) 2023 Free Software Foundation, Inc.". A note below states "This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.".

```
Dec 9 17:15
kej588406@kej588406-Vmware20:~$ javac --version
javac 21.0.9
kej588406@kej588406-Vmware20:~$ java -version
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+18-Ubuntu-124.04)
kej588406@kej588406-Vmware20:~$ gcc --version
gcc (Ubuntu 13.3.0-1ubuntu1.1) 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.
kej588406@kej588406-Vmware20:~$
```

`python3 --version`

```

kajal@kajal-OptiPlex-5090:~$ sudo apt install python3
[sudo] password for kajal:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.12.3-0ubuntu1).
python3 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
kajal@kajal-OptiPlex-5090:~$ sudo apt update
Hit:1 http://ports.ubuntu.com/ubuntu-ports main Release
Hit:2 http://ports.ubuntu.com/ubuntu-ports main-updates InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports main-backports InRelease
Hit:4 http://ports.ubuntu.com/ubuntu-ports main-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
kajal@kajal-OptiPlex-5090:~$ python3 --version
Command 'python3' not found, did you mean:
  command 'python3' from deb python3 (3.12.3-0ubuntu1)
Try: sudo apt install <deb name>
kajal@kajal-OptiPlex-5090:~$ python3 --version
Python 3.12.3
kajal@kajal-OptiPlex-5090:~$ 

```

`bash –version`

```

kajal@kajal-OptiPlex-5090:~$ python3 --version
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
kajal@kajal-OptiPlex-5090:~$ sudo apt update
Hit:1 http://ports.ubuntu.com/ubuntu-ports main Release
Hit:2 http://ports.ubuntu.com/ubuntu-ports main-updates InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports main-backports InRelease
Hit:4 http://ports.ubuntu.com/ubuntu-ports main-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
kajal@kajal-OptiPlex-5090:~$ bash --version
Command 'bash' not found, did you mean:
  command 'bash' from deb python3 (3.12.3-0ubuntu1)
Try: sudo apt install <deb name>
kajal@kajal-OptiPlex-5090:~$ bash --version
GNU bash, version 5.2.2(1)-release (searched-unknown-linux-gnu)
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU General Public License version 3 or later
.  
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
kajal@kajal-OptiPlex-5090:~$ 

```

Assignment 4.3: Compile

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

- Java and C programs need to be compiled before running, while Python and Bash programs can be run directly without compilation.

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

- C source code files are compiled into machine code and can be executed directly by the processor.

Which source code files are compiled to byte code?

- Java source code files are compiled into byte code.

Which source code files are interpreted by an interpreter?

- Python and Bash source code files are interpreted by an interpreter.

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

- C source code is expected to perform the calculation the fastest.

How do I run a Java program?

- You run a Java program by first compiling it with javac and then running it with java.

How do I run a Python program?

- You run a Python program by using the python3 command followed by the file name.

How do I run a C program?

- You run a C program by compiling it with gcc and then executing the generated file.

How do I run a Bash script?

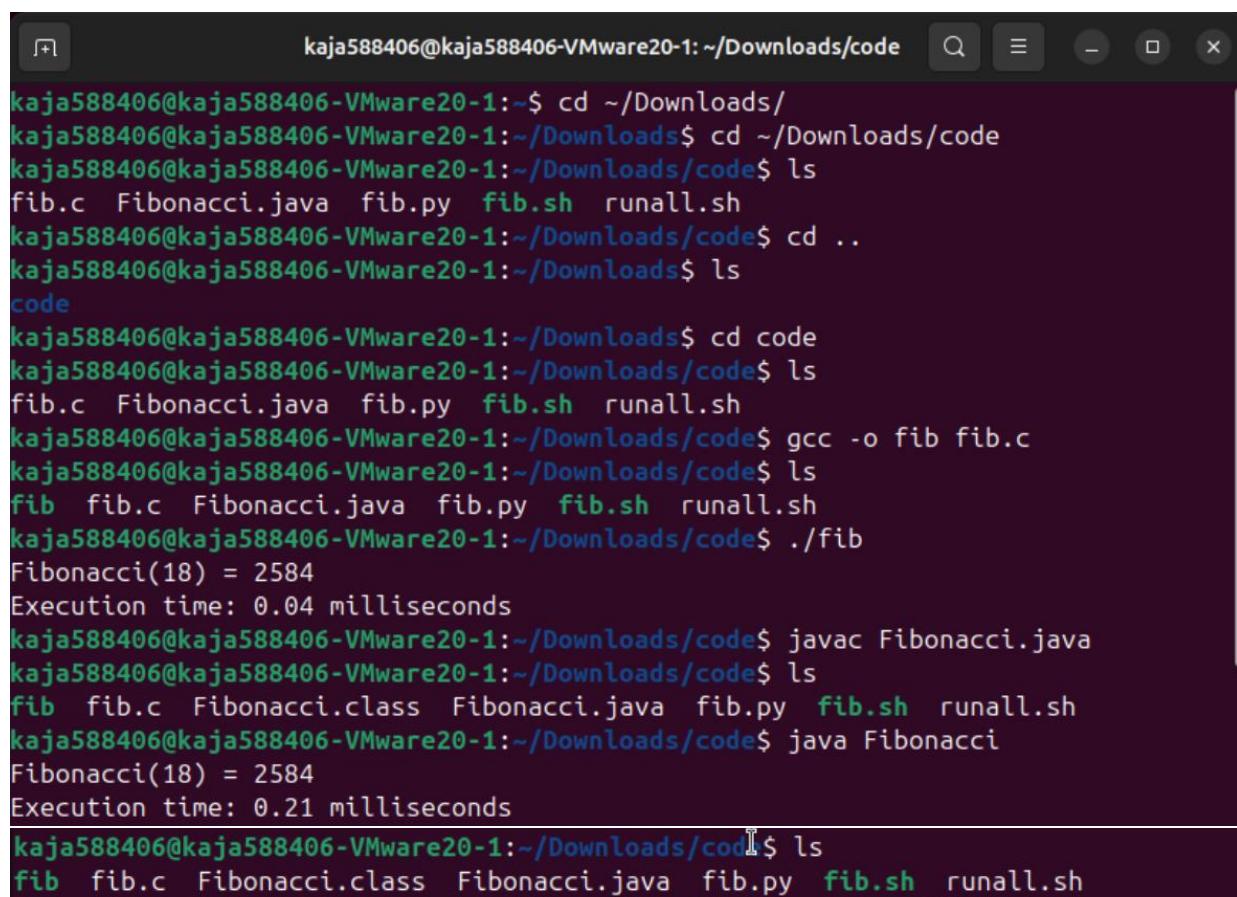
- You run a Bash script by using bash followed by the script name or by making it executable and running it directly.

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

- Yes, compiling creates a new file: Java creates a .class file, and C creates an executable file, while Python and Bash do not create a new file when run.

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?



```
kaja588406@kaja588406-VMware20-1:~/Downloads/code
kaja588406@kaja588406-VMware20-1:~/Downloads$ cd ~/Downloads/
kaja588406@kaja588406-VMware20-1:~/Downloads$ cd ~/Downloads/code
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ cd ..
kaja588406@kaja588406-VMware20-1:~/Downloads$ ls
code
kaja588406@kaja588406-VMware20-1:~/Downloads$ cd code
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ gcc -o fib fib.c
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.java fib.py fib.sh runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.04 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ javac Fibonacci.java
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.21 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
```

```

kaja588406@kaja588406-VMware20-1:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.56 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ chmod +x fib.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ chmod +x runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 1972 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo chmod a+x fib.sh
[sudo] password for kaja588406:
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 1946 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$
```

The C program is the fastest, because it runs directly as machine 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.
- Compile **fib.c** again with the optimization parameters

```

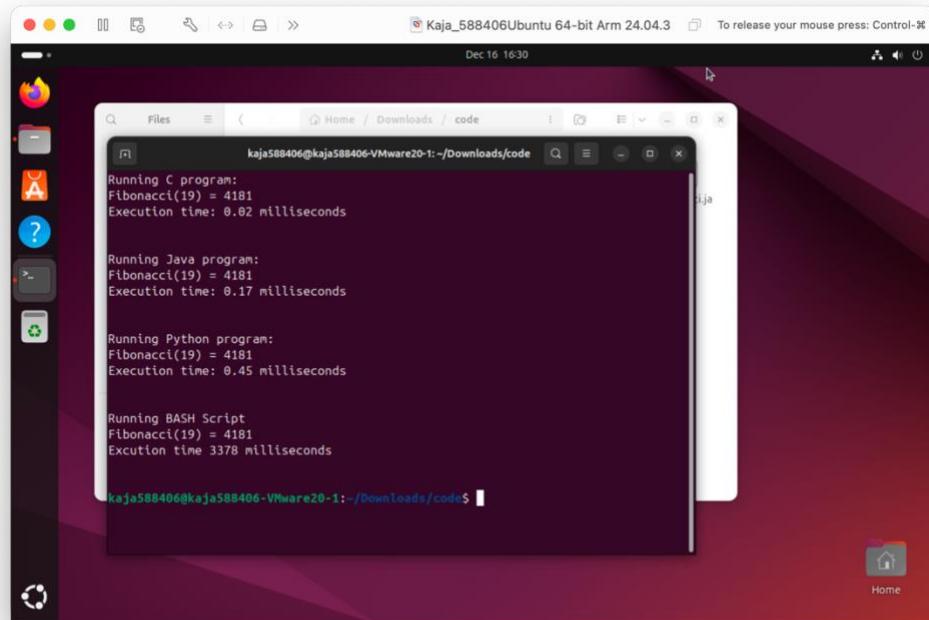
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo gcc -O2 fib.c -o fib
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$
```

- Run the newly compiled program. Is it true that it now performs the calculation faster?
Yes the calculation was 0.27 milliseconds faster after I optimised it.

```

kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo gcc -O fib.c -o fib
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.26 milliseconds
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ sudo gcc -O3 fib.c -o fib
kaja588406@kaja588406-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
```

- 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  
mov r0, #1
```

Loop:

```
mul r0, r0, r1  
subs r2, r2, #1  
cmp r2, #0  
beq End  
b Loop
```

End:

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

Screenshot of the completed code here.

The screenshot shows the OakSim simulation interface. At the top, there are browser-style controls (back, forward, refresh) and a URL bar with "wunkolo.github.io". Below the URL bar is a toolbar with icons for file operations and a "OakSim" logo. The main window has a title bar "OakSim". On the left, there is a code editor with the following assembly code:

```
1 Main:  
2     mov r1, #2  
3     mov r2, #4  
4     mov r0, #1  
5 Loop:  
6     mul r0, r0, r1  
7     sub r2, r2, #1  
8     cmp r2, #0  
9     beq End  
10    b Loop  
11 End:
```

On the right, there is a table titled "Register Value" showing the current state of the registers:

Register	Value
R0	10
R1	2
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0

Below the register table is a large memory dump area showing memory addresses from 0x000010000 to 0x000010190. The dump shows the following hex values:

```
0x000010000: 02 10 A0 E3 04 20 A0 E3 01 00  
0x000010010: 01 20 42 E2 00 00 52 E3 00 00  
0x000010020: 00 00 00 00 00 00 00 00 00 00  
0x000010030: 00 00 00 00 00 00 00 00 00 00  
0x000010040: 00 00 00 00 00 00 00 00 00 00  
0x000010050: 00 00 00 00 00 00 00 00 00 00  
0x000010060: 00 00 00 00 00 00 00 00 00 00  
0x000010070: 00 00 00 00 00 00 00 00 00 00  
0x000010080: 00 00 00 00 00 00 00 00 00 00  
0x000010090: 00 00 00 00 00 00 00 00 00 00  
0x0000100A0: 00 00 00 00 00 00 00 00 00 00  
0x0000100B0: 00 00 00 00 00 00 00 00 00 00  
0x0000100C0: 00 00 00 00 00 00 00 00 00 00  
0x0000100D0: 00 00 00 00 00 00 00 00 00 00  
0x0000100E0: 00 00 00 00 00 00 00 00 00 00  
0x0000100F0: 00 00 00 00 00 00 00 00 00 00  
0x000010100: 00 00 00 00 00 00 00 00 00 00  
0x000010110: 00 00 00 00 00 00 00 00 00 00  
0x000010120: 00 00 00 00 00 00 00 00 00 00  
0x000010130: 00 00 00 00 00 00 00 00 00 00  
0x000010140: 00 00 00 00 00 00 00 00 00 00  
0x000010150: 00 00 00 00 00 00 00 00 00 00  
0x000010160: 00 00 00 00 00 00 00 00 00 00  
0x000010170: 00 00 00 00 00 00 00 00 00 00  
0x000010180: 00 00 00 00 00 00 00 00 00 00  
0x000010190: 00 00 00 00 00 00 00 00 00 00
```

At the bottom of the interface, there is a copyright notice: "© 2017".

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