

# Template Week 4 – Software

Student number: 569073

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OaXSim ARM simulator interface. At the top, there are control buttons: 'Open', 'Run', '250', 'Step', and 'Reset'. Below these is a text area containing ARM assembly code for a factorial calculation. To the right of the code is a 'Register Value' table showing the current state of the registers. At the bottom right, there is a memory dump showing hexadecimal addresses and their corresponding values.

```
1 Main:
2     mov r2, #5
3     mov r1, #1
4
5 Loop:
6     mul r1, r1, r2
7     sub r2, r2, #1
8     cmp r2, #1
9     beq End
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
R12	0

Memory dump:

```
0x00010000: 05 20 A0 E3 01 10 00 00
0x00010010: 01 00 52 E3 00 00 00 00
0x00010020: 00 00 00 00 00 00 00 00
0x00010030: 00 00 00 00 00 00 00 00
0x00010040: 00 00 00 00 00 00 00 00
0x00010050: 00 00 00 00 00 00 00 00
```

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

The screenshot shows a terminal window titled 'jasper569073@jasperserver: ~' with a search bar and window controls. The terminal output is as follows:

```
jasper569073@jasperserver:~$ javac --version
javac 21.0.9
jasper569073@jasperserver:~$ 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)
jasper569073@jasperserver:~$ 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.
jasper569073@jasperserver:~$ python3 --version
Python 3.12.3
jasper569073@jasperserver:~$ 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.
jasper569073@jasperserver:~$ S
```

`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 are compiled files the other two are interpreted

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

Only fib.c is compiled into machine code.

Which source code files are compiled to byte code?

Fibonacci.java is compiled to byte code

Which source code files are interpreted by an interpreter?

fib.py and fib.sh are interpreted.

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

fib.c because it's a compiled language

How do I run a Java program?

Java programs run on a java virtual machine

On pc Java programs can be compiled and run using a JDK

For example "java program.java"

How do I run a Python program?

Python programs are run using the Python interpreter

For example "python program.py"

How do I run a C program?

A C program is compiled using a C compiler and then the compiled program can be executed

For example "gcc program.c -o program" then in terminal "./program"

gcc is one of the most common c compilers

How do I run a Bash script?

First you need to make the script executable then it can be run using bash

For example to make executable "chmod +x program.sh" and then "./program.sh"

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

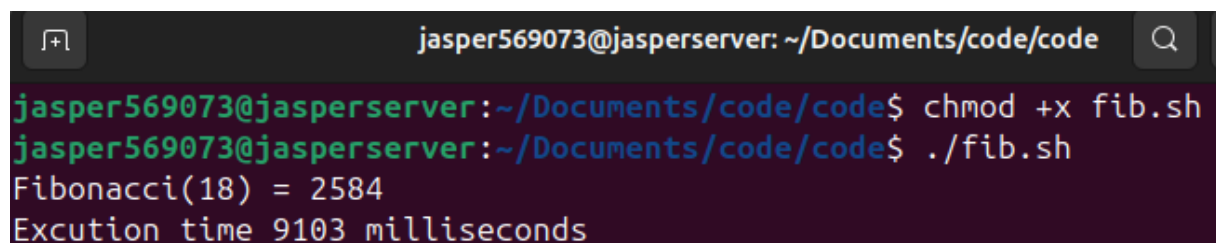
For java a new file gets created it will be a .jar file

For c a new file gets created it will be a .exe file

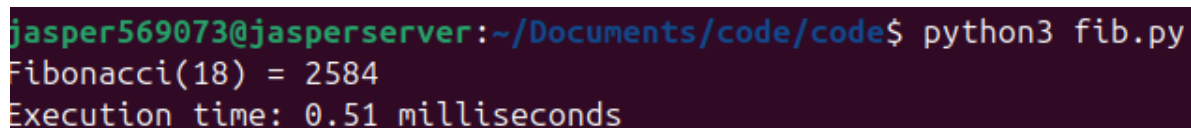
Take relevant screenshots of the following commands:

- Compile the source files where necessary
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?

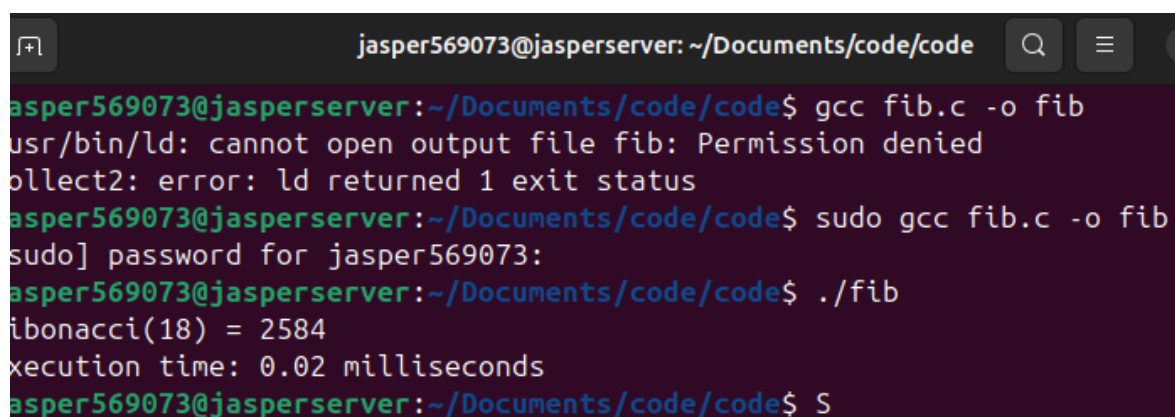
C is the fastest



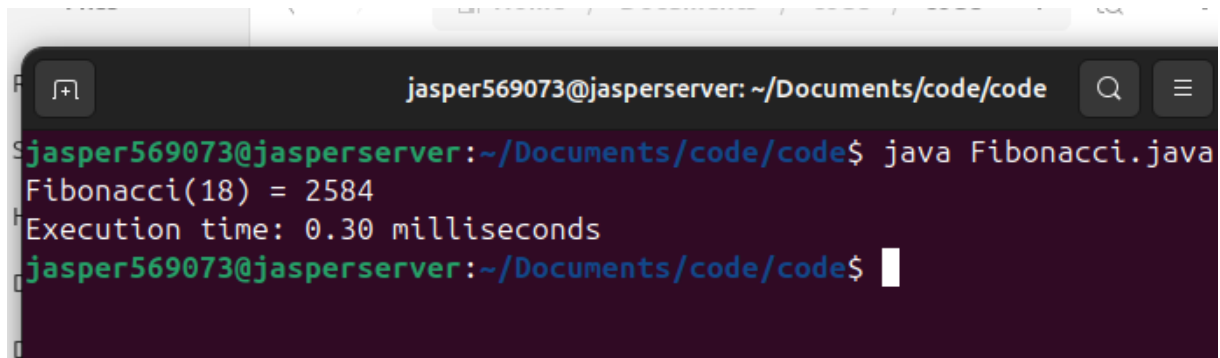
```
jasper569073@jasperserver: ~/Documents/code/code
jasper569073@jasperserver:~/Documents/code/code$ chmod +x fib.sh
jasper569073@jasperserver:~/Documents/code/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 9103 milliseconds
```



```
jasper569073@jasperserver:~/Documents/code/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.51 milliseconds
```



```
jasper569073@jasperserver: ~/Documents/code/code
jasper569073@jasperserver:~/Documents/code/code$ gcc fib.c -o fib
/usr/bin/ld: cannot open output file fib: Permission denied
collect2: error: ld returned 1 exit status
jasper569073@jasperserver:~/Documents/code/code$ sudo gcc fib.c -o fib
[sudo] password for jasper569073:
jasper569073@jasperserver:~/Documents/code/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
jasper569073@jasperserver:~/Documents/code/code$ S
```

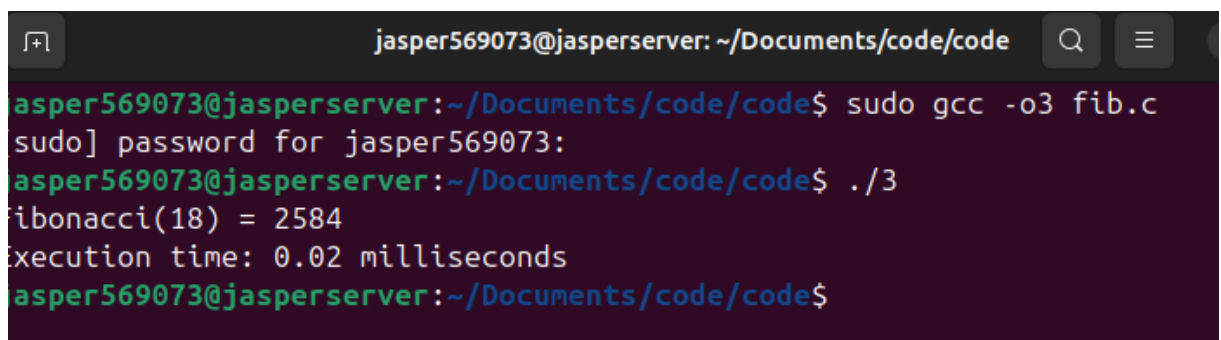
A terminal window with a dark background. The prompt is 'jasper569073@jasperserver: ~/Documents/code/code'. The user enters 'java Fibonacci.java'. The output is 'Fibonacci(18) = 2584' followed by 'Execution time: 0.30 milliseconds'. The prompt returns.

```
jasper569073@jasperserver: ~/Documents/code/code$ java Fibonacci.java
Fibonacci(18) = 2584
Execution time: 0.30 milliseconds
jasper569073@jasperserver: ~/Documents/code/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.  
-o3
- Compile **fib.c** again with the optimization parameters

A terminal window with a dark background. The prompt is 'jasper569073@jasperserver: ~/Documents/code/code'. The user enters 'sudo gcc -o3 fib.c'. The prompt changes to 'sudo] password for jasper569073:'. The user enters a password. The prompt returns to 'jasper569073@jasperserver: ~/Documents/code/code'. The user enters './3'. The output is 'fibonacci(18) = 2584' followed by 'Execution time: 0.02 milliseconds'. The prompt returns.

```
jasper569073@jasperserver: ~/Documents/code/code$ sudo gcc -o3 fib.c
sudo] password for jasper569073:
jasper569073@jasperserver: ~/Documents/code/code$ ./3
fibonacci(18) = 2584
Execution time: 0.02 milliseconds
jasper569073@jasperserver: ~/Documents/code/code$
```

- Run the newly compiled program. Is it true that it now performs the calculation faster?  
It's the exact same time so it doesn't seem faster with this program
- 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.

```
jasper569073@jasperserver: ~/Documents/code/code
GNU nano 7.2 runall.sh
#!/bin/bash
clear
n=19

echo "Running C program:"
./fib $n
echo -e '\n'

echo "Running Java program:"
java Fibonacci.java $n
echo -e '\n'

echo "Running Python program:"
python3 fib.py $n
echo -e '\n'

echo "Running BASH Script"
./fib.sh $n
echo -e '\n'
```

e)

```
jasper569073@jasperserver: ~/Documents/code/cod

Running C program:
Fibonacci(19) = 4181
Execution time: 0.03 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.49 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.95 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 14262 milliseconds

jasper569073@jasperserver:~/Documents/code/code$ S
```

#### 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
```

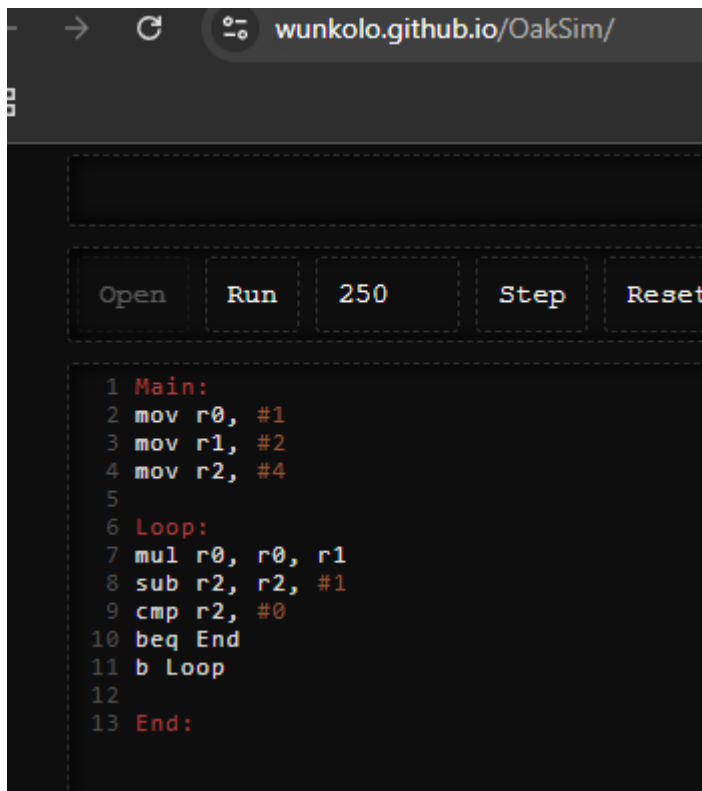
```
mov r2, #4
```

Loop:

End:

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

Screenshot of the completed code here.

A screenshot of a web-based ARM assembler simulator. The browser address bar shows 'wunkolo.github.io/OakSim/'. Below the address bar is a control panel with buttons for 'Open', 'Run', '250' (likely a counter or value), 'Step', and 'Reset'. The main area displays ARM assembly code with line numbers 1 through 13. The code implements a loop to calculate 2^4 by multiplying r0 by r1 four times, decrementing r2 until it reaches 0, and then jumping to the End label.

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

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