

# Template Week 4 – Software

Student number: 591658

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

Factorial of 5

The screenshot shows an ARM assembly simulator interface. At the top, there are buttons for 'Open', 'Run', '250', 'Step', and 'Reset'. The 'Run' button is highlighted. Below the buttons is a text area containing the following assembly code:

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

To the right of the code is a 'Register Value' table:

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

Below the register table is a memory dump showing hexadecimal addresses and their corresponding values in hexadecimal and decimal:

```
0x00010000: 05 20 A0 E3 02 10 A0 E1 01 20 42 E2 91 03
0x00010010: 01 00 52 E3 00 00 00 0A FA FF FF EA 00 00
0x00010020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

## Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

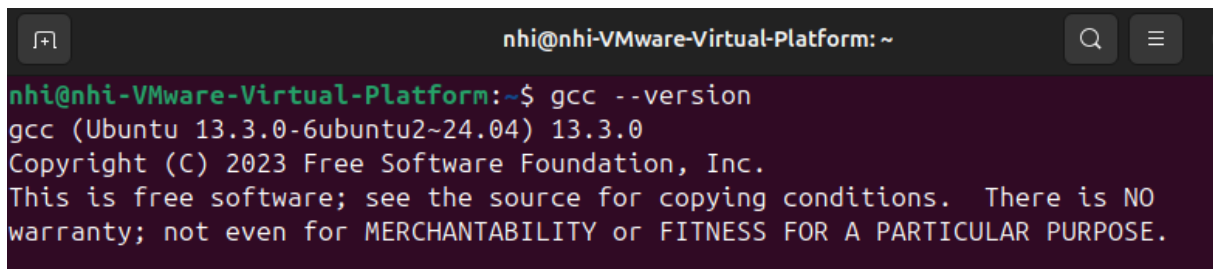
The screenshot shows a terminal window with the prompt 'nhi@nhi-VMware-Virtual-Platform: ~'. The command 'javac --version' has been entered and executed, resulting in the output 'javac 21.0.9'.

java --version

The screenshot shows a terminal window with the prompt 'nhi@nhi-VMware-Virtual-Platform: ~'. The command 'java --version' has been entered and executed, resulting in the output:

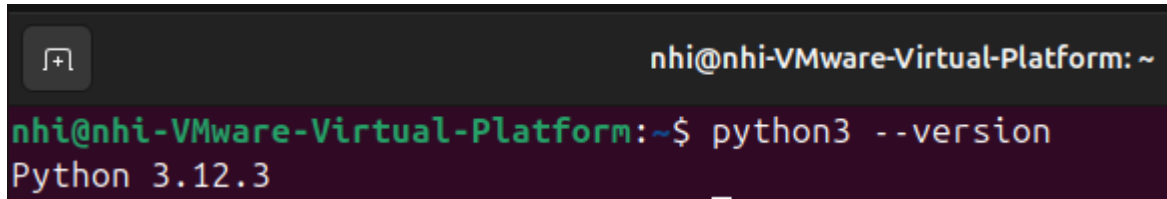
```
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
nhi@nhi-VMware-Virtual-Platform:~$
```

gcc --version

A terminal window titled 'nhi@nhi-VMware-Virtual-Platform: ~' with search and menu icons in the top right. The command 'gcc --version' has been executed, resulting in the following output:

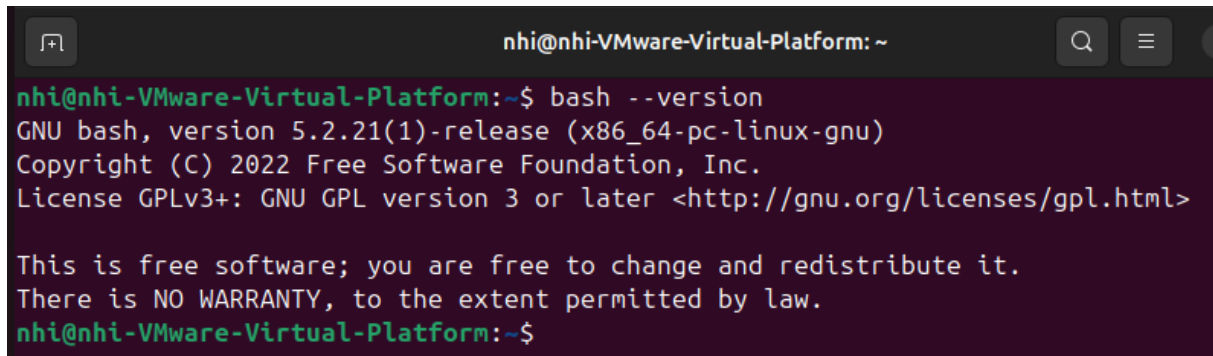
```
nhi@nhi-VMware-Virtual-Platform:~$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

python3 --version

A terminal window titled 'nhi@nhi-VMware-Virtual-Platform: ~' with search and menu icons in the top right. The command 'python3 --version' has been executed, resulting in the following output:

```
nhi@nhi-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
```

bash --version

A terminal window titled 'nhi@nhi-VMware-Virtual-Platform: ~' with search and menu icons in the top right. The command 'bash --version' has been executed, resulting in the following output:

```
nhi@nhi-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.
nhi@nhi-VMware-Virtual-Platform:~$
```

### **Assignment 4.3: Compile**

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

fib.c

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

fib.sh

Which source code files are compiled to byte code?

Fibonacci.java

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?

fib.c

#### **How do I run a Java program?**

Install java and javac compiler

Use javac compiler to compile Java file into a bytecode file (.class file) with the same name in the same directory

Command: javac Fibonacci.java

Run the bytecode file by command: java Fibonacci

#### **How do I run a Python program?**

Using python interpreter python3.

Command: python3 fib.py

#### **How do I run a C program?**

Use C compiler gcc to compile the file into an executable file.

Command: gcc fib.c -o fib

Run the executable file by command: ./fib

#### **How do I run a Bash script?**

Make Bash script file executable by command: sudo chmod a+x fib.sh

Run the file by command: sudo ./fib.sh

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

To run java source file, a bytecode (.class) file will be created

To run c source code file, a compiled file will be created

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 file run fastest

Running Fibonacci.java

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.55 milliseconds
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$
```

Running fib.c

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -o fib
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$
```

Running fib.py

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.60 milliseconds
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$
```

Running fib.sh

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x fib.sh
[sudo] password for nhi:
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Execution time 9195 milliseconds
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$
```

Running runall.sh to find out which source code file performs the calculation fastest

```
nhi@nhi-VMware-Virtual-Platform: ~/Downloads/code

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

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.57 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.84 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 14588 milliseconds
```

## Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

- a) 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.

-O1 Optimize.

-O2 Optimize even more.

-O3 Optimize yet more.

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

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -O1 -o fib1
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -O2 -o fib2
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib fib1 fib2 fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
```

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

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -O1 -o fib1
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -O2 -o fib2
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib fib1 fib2 fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ./fib1
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ./fib2
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
```

- 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.

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls -l runall.sh
-rw-rw-r-- 1 nhi nhi 249 Jun  9 2023 runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x runall.sh
[sudo] password for nhi:
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls -l runall.sh
-rwxrwxr-x 1 nhi nhi 249 Jun  9 2023 runall.sh
```

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
nhi@nhi-VMware-Virtual-Platform:~/Downloads/code$ ./runall.sh
```

```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds
```

```
Running Java program:
Fibonacci(19) = 4181
Execution time: 0.38 milliseconds
```

```
Running Python program:
Fibonacci(19) = 4181
Execution time: 1.29 milliseconds
```

```
Running BASH Script
Fibonacci(19) = 4181
Execution time 14617 milliseconds
```

```
nhi@nhi-VMware-Virtual-Platform:~/Downloads/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.

Open
Run
250
Step
Reset

```

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

```

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

0x00010000:	02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 00 00
0x00010010:	03 00 00 0A 90 01 00 E0 01 20 42 E2 00 00
0x00010020:	F9 FF FF EA 00 00 00 00 00 00 00 00 00 00
0x00010030:	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010040:	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010050:	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010060:	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010070:	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010080:	00 00 00 00 00 00 00 00 00 00 00 00 00 00

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