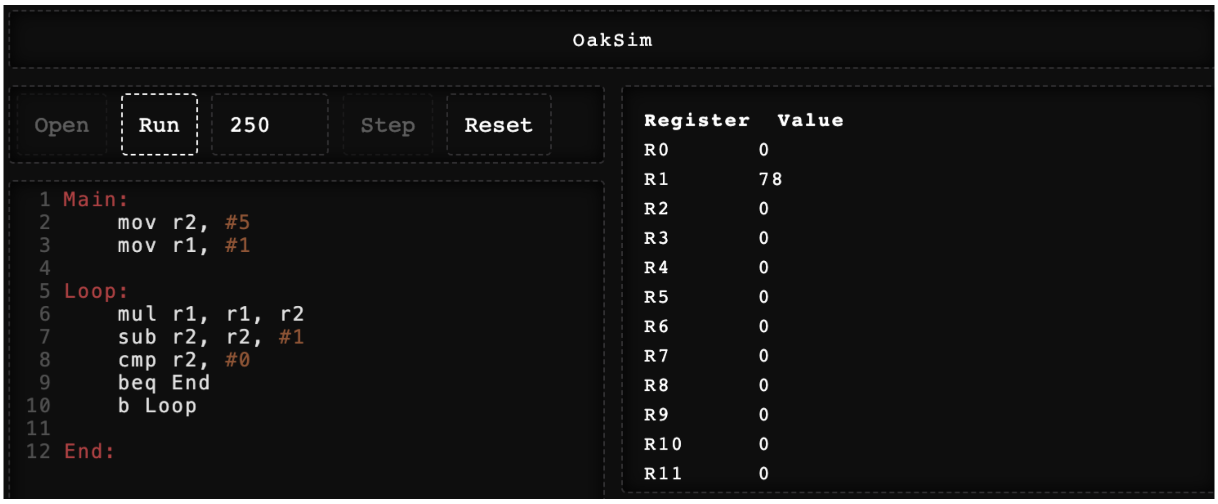
Template Week 4 – Software

Student number: 590190

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
    mov r2, #5
    mov r1, #1

Loop:
    mul r1, r1, r2
    sub r2, r2, #1
    cmp r2, #0
    beq End
    b Loop

End:
```

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

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

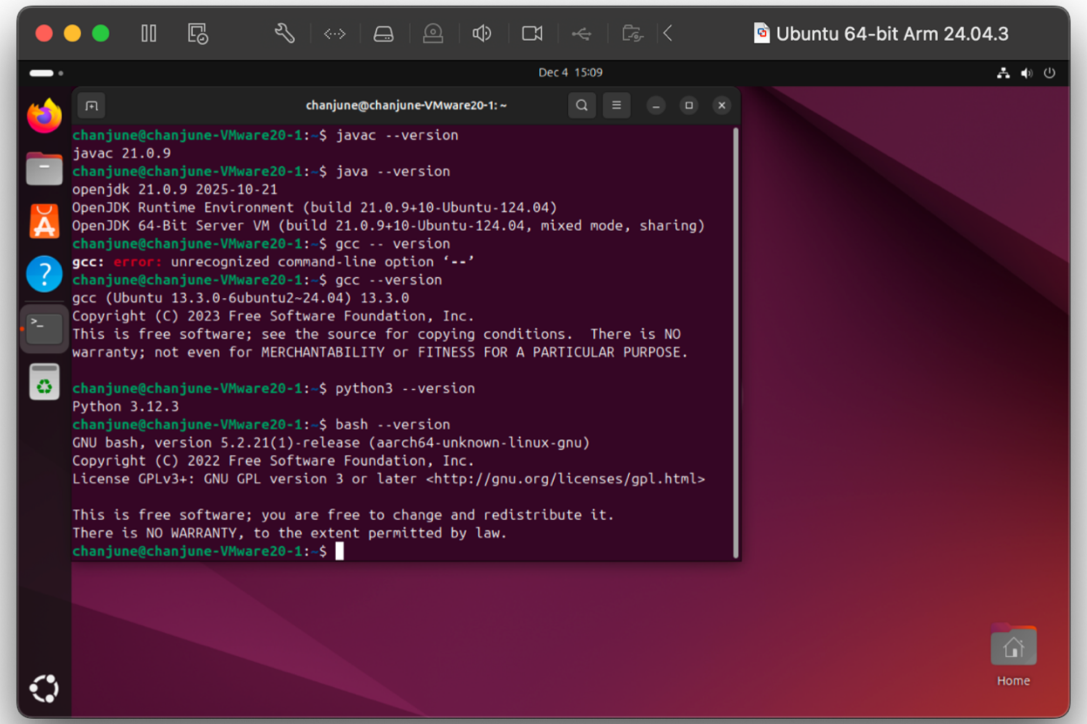
javac --version

java --version

gcc --version

python3 --version

bash --version



```
chanjune@chanjune-VMware20-1:~$ javac --version
javac 21.0.9
chanjune@chanjune-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)
chanjune@chanjune-VMware20-1:~$ gcc -- version
gcc: error: unrecognized command-line option '--'
chanjune@chanjune-VMware20-1:~$ 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.

chanjune@chanjune-VMware20-1:~$ python3 --version
Python 3.12.3
chanjune@chanjune-VMware20-1:~$ bash --version
GNU bash, version 5.2.21(1)-release (aarch64-unknown-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.
chanjune@chanjune-VMware20-1:~$
```

1. Javac: 21.0.9
2. Java: openjdk 21.0.9
3. Gcc: 13.3.0
4. Python3: 3.12.3
5. Bash: 5.2.21

Assignment 4.3: Compile

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

- fib.c, Fibonacci.java

- Which source code files are compiled into machine code and are 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, fib.sh

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

- The C program (fib.c) is expected to be the fastest. This is because C is compiled directly into machine code, making it the closest to the hardware and most efficient for execution.

- How do I run a Java program?

- Compile it: javac Fibonacci.java -> Run it: java Fibonacci

- How do I run a Python program?

- Run command: python3 fib.py

- How do I run a C program?

- Compile it: gcc fib.c -0 fib_c -> Run it: ./fib_c

- How do I Run a Bash Script?

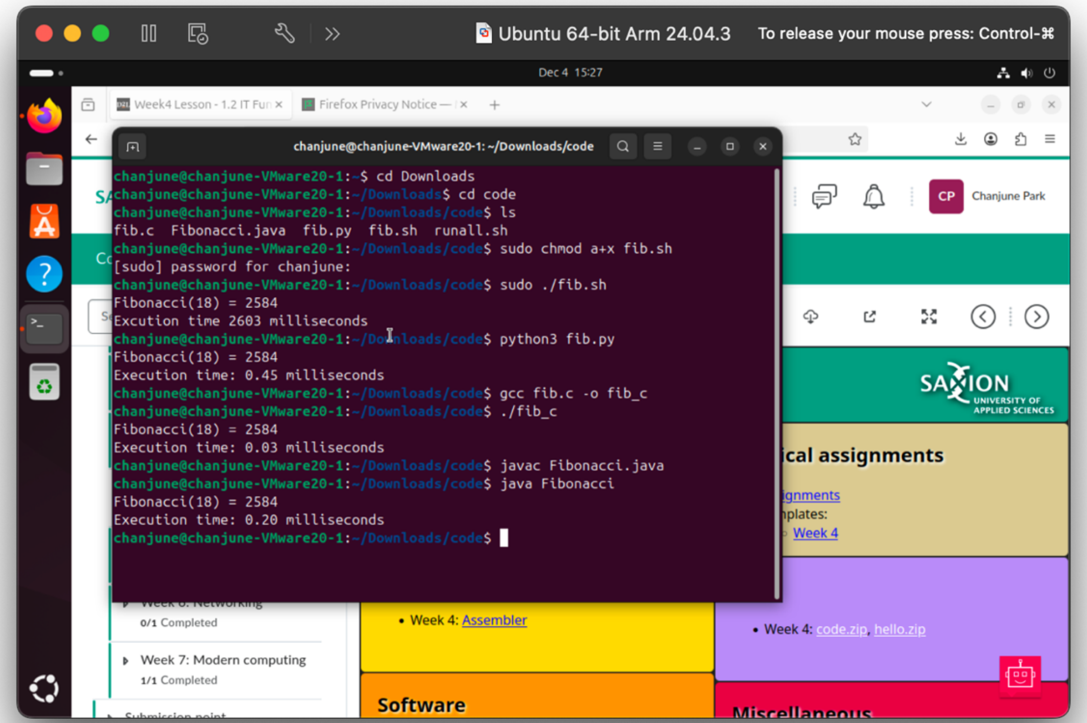
- Make it executable: sudo chmod a+x fib.sh -> sudo ./fib.sh

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

- Yes, new files are created for the compiled languages: For Java: A Fibonacci.class file (byte code) is created. For C: An executable file (e.g., fib_c or a.out) is 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?



```
chanjune@chanjune-VMware20-1:~$ cd Downloads
chanjune@chanjune-VMware20-1:~/Downloads$ cd code
chanjune@chanjune-VMware20-1:~/Downloads/code$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
chanjune@chanjune-VMware20-1:~/Downloads/code$ sudo chmod a+x fib.sh
[sudo] password for chanjune:
chanjune@chanjune-VMware20-1:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Excution time 2603 milliseconds
chanjune@chanjune-VMware20-1:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.45 milliseconds
chanjune@chanjune-VMware20-1:~/Downloads/code$ gcc fib.c -o fib_c
chanjune@chanjune-VMware20-1:~/Downloads/code$ ./fib_c
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
chanjune@chanjune-VMware20-1:~/Downloads/code$ javac Fibonacci.java
chanjune@chanjune-VMware20-1:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.20 milliseconds
chanjune@chanjune-VMware20-1:~/Downloads/code$
```

- The C program: 0.03ms (First)

- Java: 0.20ms (Second)

- Python: 0.45ms (Third)

- Bash: 2603ms (Fourth)

The C program (fib.c) performed the calculation the fastest, taking only 0.03 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.

- The parameter is `-O` (specifically `-O3` for high optimization).

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

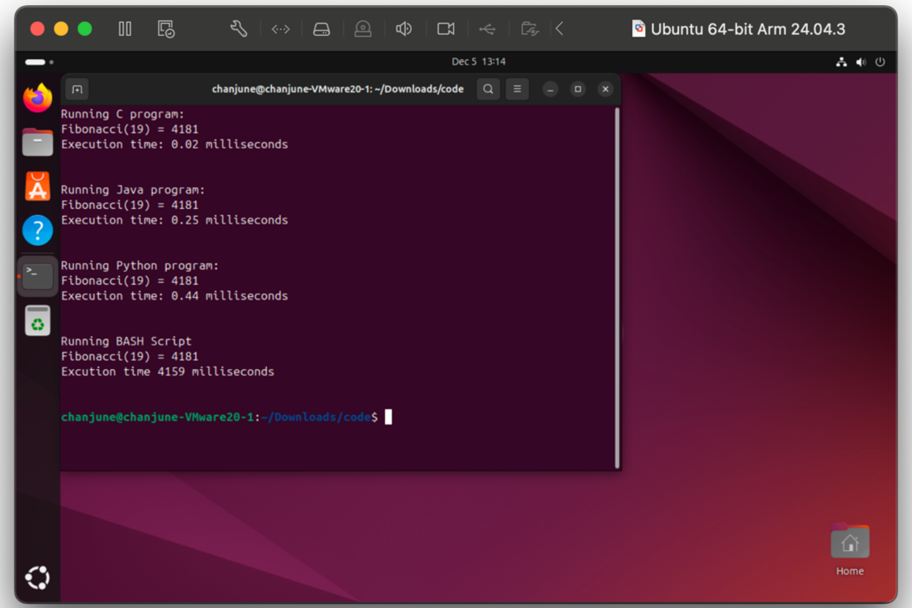
- Yes, compiled using : gcc -O3 fib.c -o fib

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

- ./runall.sh
- Yes, the execution time typically decreases because the -O3 flag enables aggressive compiler optimizations, making the machine code more efficient.

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.

- I modified the runall.sh script to execute the compiled C program (fib) along with the Java, Python, and Bash versions sequentially. When I ran the script, all four programs executed one after another as expected.


Ubuntu 64-bit Arm 24.04.3
Dec 5 13:14
chanjune@chanjune-VMware20-1: ~/Downloads/code
Home

```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.02 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.25 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.44 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 4159 milliseconds


chanjune@chanjune-VMware20-1:~/Downloads/code$
```

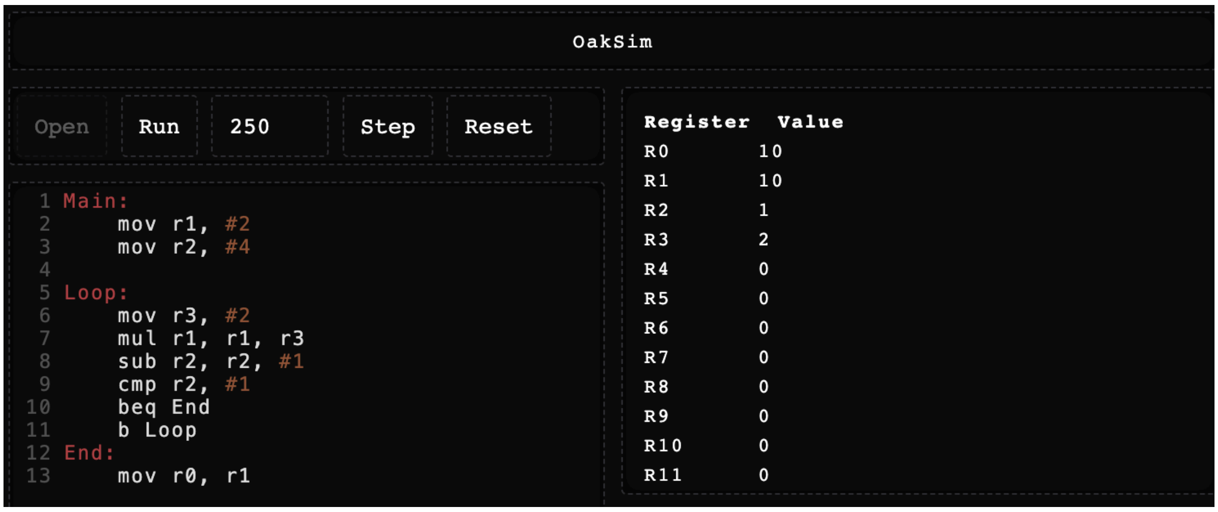
Assignment 4.5: More ARM Assembly

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:

      mov r3, #2

      mul r1, r1, r3

    sub r2, r2, #1

    cmp r2, #1

    beq End

    b Loop

End:

      mov r0, r1
```

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

Screenshot of the completed code here.



```
Main:
    mov r1, #2
    mov r2, #4

Loop:
    mov r3, #2
    mul r1, r1, r3
    sub r2, r2, #1
    cmp r2, #1
    beq End
    b Loop
End:
    mov r0, r1
```

```
Register  Value
R0        10
R1        10
R2        1
R3        2
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
R11       0
```

Hexadecimal 10 is equivalent to decimal 16.

Ready? Save this file and export it as a pdf file with the name: **week4.pdf**