

Template Week 4 – Software

Student number: 585303

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim software interface. On the left, there is a text editor window containing ARM assembly code. On the right, there is a register dump and memory dump window.

Assembly Code:

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

Registers:

| Register | Value |
|----------|-------|
| R0 | 0 |
| R1 | 78 |
| R2 | 1 |
| R3 | 0 |
| R4 | 0 |

Memory Dump:

| Address | Value |
|------------|--|
| 0x00010000 | 05 20 A0 E3 02 10 A0 E1 01 20 42 |
| 0x00010010 | 01 00 52 E3 00 00 00 00 0A FA FF FF |
| 0x00010020 | 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 |
| 0x00010040 | 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 |
| 0x00010060 | 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 |
| 0x00010080 | 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x00010090 | 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x000100A0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x000100B0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x000100C0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x000100D0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x000100E0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x000100F0 | 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, java --version, gcc --version, python3 --version, bash --version

The screenshot shows a terminal window with the following command outputs:

```
sabrina585303@sabrina585303-VMware20-1:~$ python3 --version
Python 3.12.3
sabrina585303@sabrina585303-VMware20-1:~$ javac --version
javac 21.0.9
sabrina585303@sabrina585303-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)
sabrina585303@sabrina585303-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.

sabrina585303@sabrina585303-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.
sabrina585303@sabrina585303-VMware20-1:~$
```

Assignment 4.3: Compile

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

- Fibonacci.java
- 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
- Fib.sh

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

- Fib.c will be the fastest because it is being translated into machine code by a compiler, which makes it able to be executed directly by the processor.

How do I run a Java program?

- Javac Fibonacci.java
- Java Fibonacci

How do I run a Python program?

- Python3 fib.py

How do I run a C program?

- gcc fib.c -o fib
- ./fib

How do I run a Bash script?

- chmod +x fib.sh
- ./fib.sh

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

- Fibonacci.class for Fibonacci.java
- An executable file for fib.c and we name it fib

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?: the fib.c

```
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ chmod +x fib.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 3632 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.64 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
sabrina585303@sabrina585303-VMware20-1:~/Downloads/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.
 - I will use the parameter **gcc: -O3**.
 - It is a strong optimization level that tries to maximize the speed of the compiled program.
 - **gcc -O3 fib.c -o fib**

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

```
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ man gcc
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ gcc -O3 fib.c -o fib
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ time ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds

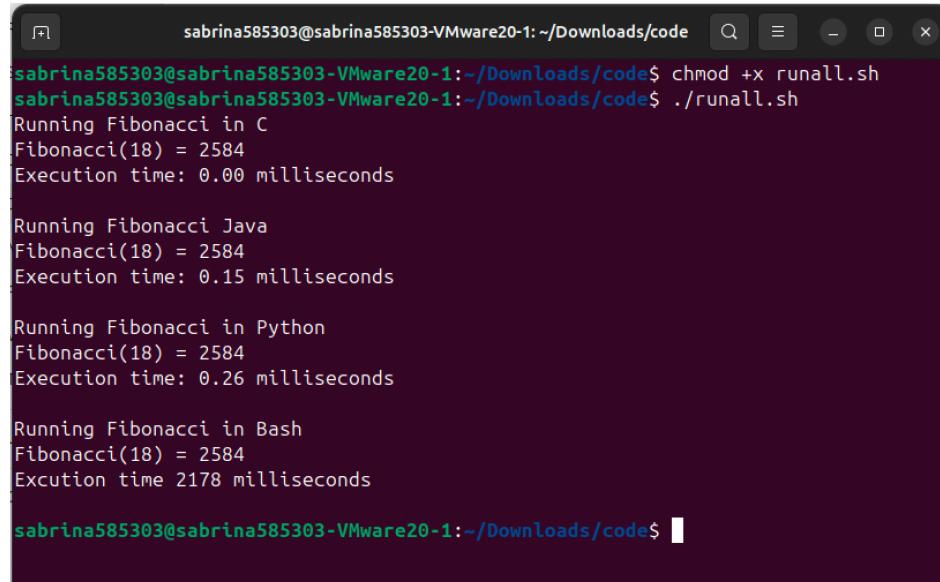
real    0m0.003s
user    0m0.000s
sys     0m0.003s
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$
```

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

```
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ man gcc
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ gcc -O3 fib.c -o fib
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ time ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.001s
sys     0m0.001s
sabrina585303@sabrina585303-VMware20-1:~/Downloads/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.



```

sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ chmod +x runall.sh
sabrina585303@sabrina585303-VMware20-1:~/Downloads/code$ ./runall.sh
Running Fibonacci in C
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds

Running Fibonacci Java
Fibonacci(18) = 2584
Execution time: 0.15 milliseconds

Running Fibonacci in Python
Fibonacci(18) = 2584
Execution time: 0.26 milliseconds

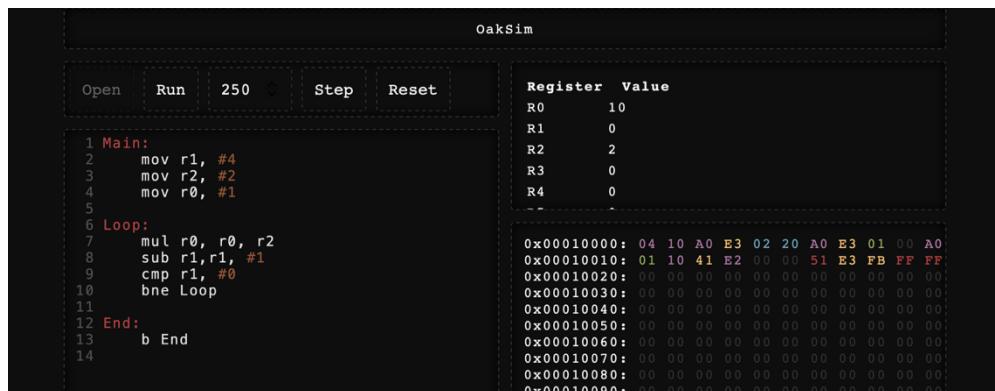
Running Fibonacci in Bash
Fibonacci(18) = 2584
Excution time 2178 milliseconds

sabrina585303@sabrina585303-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:

```

1 Main:
2     mov r1, #4
3     mov r2, #2
4     mov r0, #1
5
6 Loop:
7     mul r0, r0, r2
8     sub r1, r1, #1
9     cmp r1, #0
10    bne Loop
11
12 End:
13    b End
14

```

| Register | Value |
|----------|-------|
| R0 | 10 |
| R1 | 0 |
| R2 | 2 |
| R3 | 0 |
| R4 | 0 |

Memory Dump:

| Address | Value |
|-------------|---|
| 0x00010000: | 04 10 A0 E3 02 20 A0 E3 01 00 A0 |
| 0x00010010: | 01 10 41 E2 00 00 51 E3 FB FF FF |
| 0x00010020: | 00 00 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 00 00 |
| 0x00010040: | 00 00 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 00 00 |
| 0x00010060: | 00 00 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 00 00 |
| 0x00010080: | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x00010090: | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |