

Template Week 4 – Software

Student number: 578634

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a software interface for assembly code. At the top, there are buttons for 'Open', 'Run' (highlighted), 'Step', and 'Reset'. A dropdown menu is set to '250'. The assembly code is as follows:

```
1 Main:  
2     mov r1, #1  
3     mov r2, #5  
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:  
13 b End
```

At the bottom, a stack trace is displayed:

```
abort() at jsStackTrace@https://wunkolo.github.io/OakSim/lib/  
unicorn-arm.min.js:5:18821 stackTrace@https://wunkolo.github.io/  
OakSim/lib/unicorn-arm.min.js:5:18992 abort@https://  
wunkolo.github.io/OakSim/lib/unicorn-arm.min.js:29:7211  
_abort@https://wunkolo.github.io/OakSim/lib/unicorn-  
arm.min.js:5:200315 sb@https://wunkolo.github.io/OakSim/lib/unicorn-  
arm.min.js:16:17608 TMa@https://wunkolo.github.io/OakSim/lib/  
unicorn-arm.min.js:9:170658 invoke_iii@https://wunkolo.github.io/
```

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
0x00010000:	01 10 A0 E3 05 20 A0 E3 91 02 01 E0 01 20 42 E2 ..
0x00010010:	01 00 52 E3 00 00 00 0A FA FF FF EA FE FF FF EA .. R
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 ..

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

gcc --version

python3 --version

bash --version

```
konstantin@konstantin-VMware-Virtual-Platform:~$ javac --version
javac 21.0.8
konstantin@konstantin-VMware-Virtual-Platform:~$ java --version
openjdk 21.0.8 2025-07-15
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu124.04.1, mixed mode, sha
ring)
konstantin@konstantin-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.

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

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?

- Fastest code file will be fib.c due to it 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 ./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 named 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?

```
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java

konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.22 milliseconds
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.30 milliseconds
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -o fib
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ chmod a+x fib.py
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ ./fib.py
./fib.py: line 1: import: command not found
./fib.py: line 2: import: command not found
./fib.py: line 4: syntax error near unexpected token `(
./fib.py: line 4: `def fibonacci(n):'
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ sudo chmod a+x fib.sh
[sudo] password for konstantin:
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ sudo ./fib.sh
Fibonacci(18) = 2584
Execution time 5894 milliseconds
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ █
```

The fastest source code is the C file.

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.
 - a. Parameter found in man gcc: -O2
 - b. Explanation: enables a higher level of optimization, making the compiled program run faster.
 - c. **gcc -O2 fib.c -o fib**
- Compile **fib.c** again with the optimization parameters

```
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ man gcc
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ gcc -O2 fib.c -o fib
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ time ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.001s
sys     0m0.001s
```

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

```
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ man gcc
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ gcc -O2 fib.c -o fib
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ time ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.001s
sys     0m0.001s
```

```
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ gcc fib.c -o fib
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
konstantin@konstantin-VMware-Virtual-Platform:~/Downloads/code$ chmod +x fib.py
```

Using the parameters makes the execution time faster by 0.01 millisecond

- 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.

The screenshot shows a terminal window with the following output:

```
konstantin@konstantin-Virtual-Platform: ~/Downloads/code
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.27 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.45 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Excution time 10053 milliseconds

konstantin@konstantin-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.

oakSim

Open Run 250 Step Reset

```
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 Exit:
13    b Exit
```

Register	Value
R0	2
R1	3
R2	2
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0

0x00010000: 04 10 A0 E3 02 20 A0 E3 01 00 A0 E3 90
0x00010010: 01 10 41 E2 00 00 51 E3 FB FF FF 1A FE
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
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
0x00010090: 00 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 00
0x000100B0: 00 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 00
0x000100D0: 00 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 00
0x000100F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010120: 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](#)