

# Template Week 4 – Software

Student number: 588963

## Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a web-based ARM assembler simulator. The main window displays the following assembly code:

```
1 Main: mov r1, #1 //588963 Tony Jarwa
2      mov r2, #5
3
4 Loop: mul r1, r1, r2
5      sub r2, r2, #1
6      cmp r2, #1
7      bcc End
8      b Loop
9
10
11
12
```

On the right, the Register Value table shows:

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

Below the register table, the memory dump shows the execution state, with the value 78 (0x4E) in R1 and 1 (0x01) in R2.

78 in Hex is 120 in Decimal.

## Assignment 4.2: Programming languages

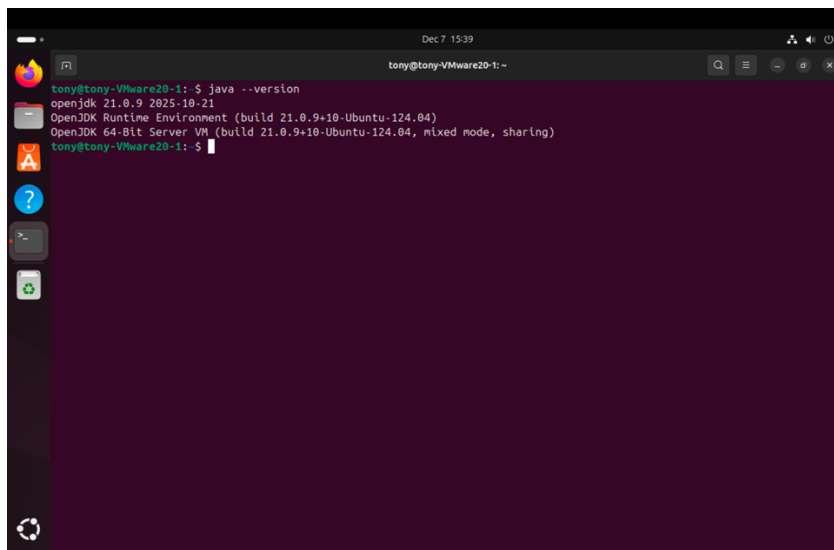
Take screenshots that the following commands work:

javac -version

The screenshot shows a terminal window with the following output:

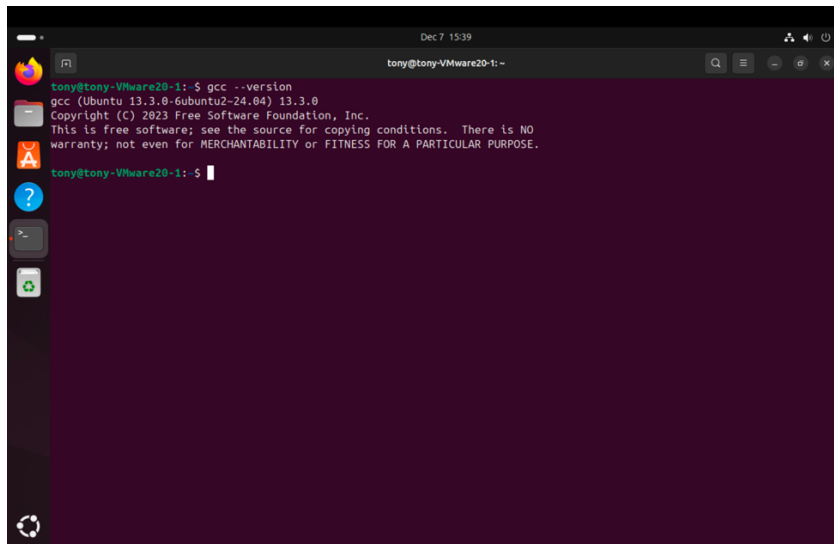
```
tony@tony-VMware20-1:~$ javac --version
javac 21.0.9
tony@tony-VMware20-1:~$
```

java -version



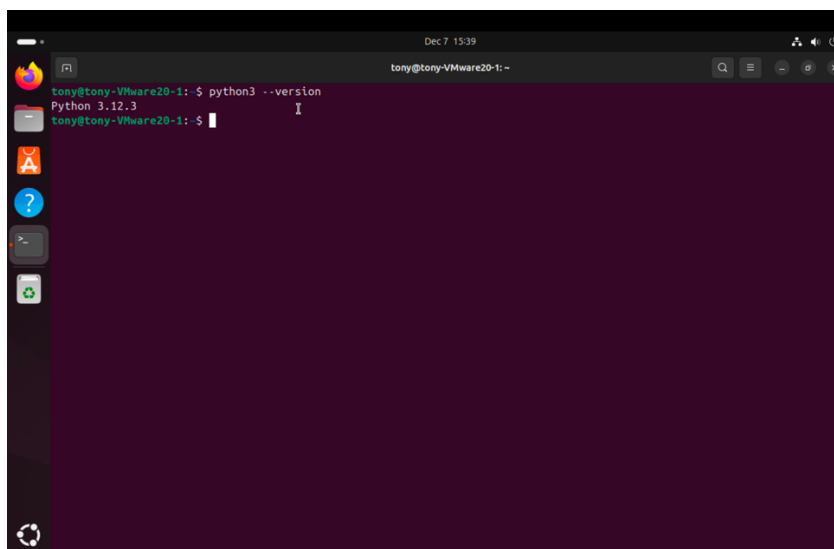
```
Dec 7 15:39
tony@tony-VMware20-1: ~
tony@tony-VMware20-1:~$ java -version
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
tony@tony-VMware20-1:~$
```

gcc -version



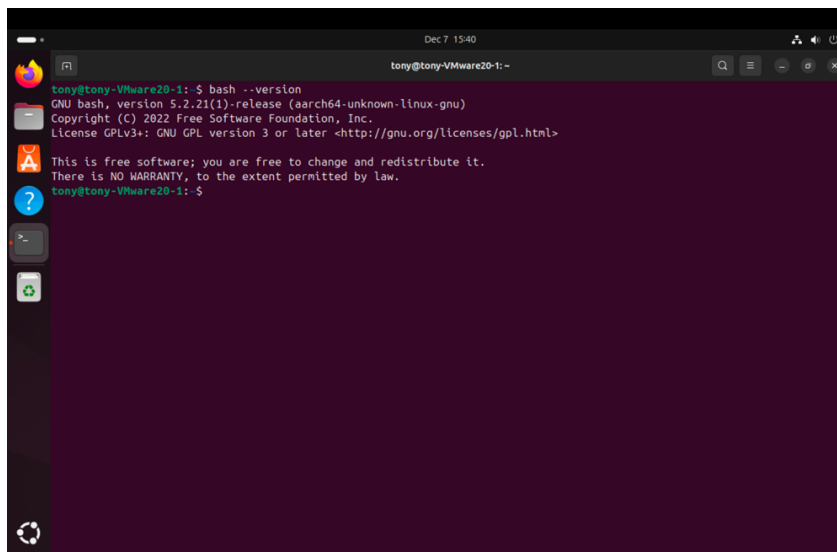
```
Dec 7 15:39
tony@tony-VMware20-1: ~
tony@tony-VMware20-1:~$ gcc -version
gcc (Ubuntu 13.3.0-6ubuntu2-24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
tony@tony-VMware20-1:~$
```

python3 -version



```
Dec 7 15:39
tony@tony-VMware20-1: ~
tony@tony-VMware20-1:~$ python3 --version
Python 3.12.3
tony@tony-VMware20-1:~$
```

bash --version



```
Dec 7 15:40
tony@tony-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.
tony@tony-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?

The fib.c (C program) is expected to be the fastest one because it is compiled directly into machine code.

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 a+x fib.sh
```

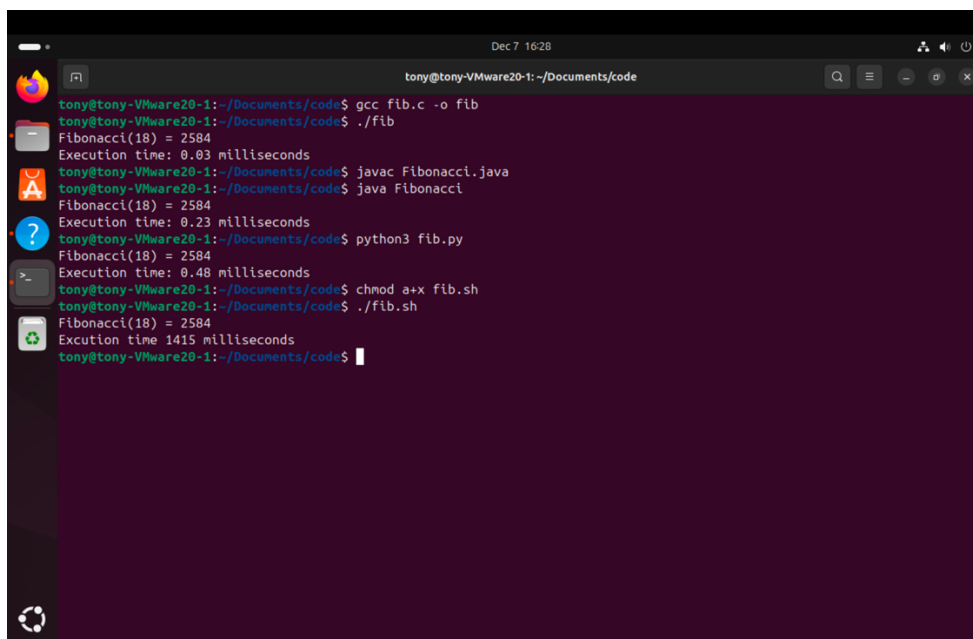
```
./fib.sh
```

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

Yes, by Fibonacci.java a new file (Fibonacci.class), and fib.c (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?



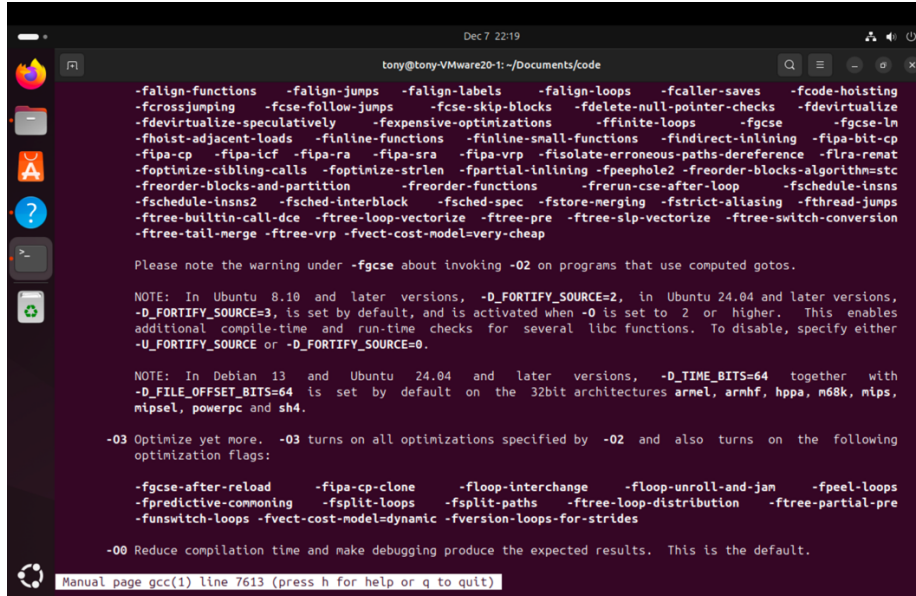
```
Dec 7 16:28
tony@tony-VMware20-1: ~/Documents/code
tony@tony-VMware20-1:~/Documents/code$ gcc fib.c -o fib
tony@tony-VMware20-1:~/Documents/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.03 milliseconds
tony@tony-VMware20-1:~/Documents/code$ javac Fibonacci.java
tony@tony-VMware20-1:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.23 milliseconds
tony@tony-VMware20-1:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.48 milliseconds
tony@tony-VMware20-1:~/Documents/code$ chmod a+x fib.sh
tony@tony-VMware20-1:~/Documents/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 1415 milliseconds
tony@tony-VMware20-1:~/Documents/code$
```

The C program (fib.c) is the fastest (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.



```
Dec 7 22:19
tony@tony-VMware20-1: ~/Documents/code

-falign-functions -falign-jumps -falign-labels -falign-loops -fcallee-saves -fcode-hoisting
-fcrossjumping -fcse-follow-jumps -fcse-skip-blocks -fdelete-null-pointer-checks -fdevirtualize
-fdevirtualize-speculatively -fexpensive-optimizations -ffinite-loops -fgcse -fgcse-lin
-fhoist-adjacent-loads -finline-functions -finline-small-functions -findirect-inlining -fipa-bit-cp
-fipa-cp -fipa-icf -fipa-ra -fipa-sra -fipa-vrp -fisolate-erroneous-paths-dereference -flra-renat
-foptimize-sibling-calls -foptimize-strlen -fpartial-inlining -fpeephole2 -freorder-blocks-algorithm=stc
-freorder-blocks-and-partition -freorder-functions -frerun-cse-after-loop -fschedule-insns
-fschedule-insns2 -fsched-interblock -fsched-spec -fstore-merging -fstrict-aliasing -fthread-jumps
-ftree-builtin-call-dce -ftree-loop-vectorize -ftree-slp-vectorize -ftree-switch-conversion
-ftree-tail-merge -ftree-vrp -fvect-cost-model=very-cheap

Please note the warning under -fgcse about invoking -O2 on programs that use computed gotos.

NOTE: In Ubuntu 8.10 and later versions, -D_FORTIFY_SOURCE=2, in Ubuntu 24.04 and later versions,
-D_FORTIFY_SOURCE=3, is set by default, and is activated when -O is set to 2 or higher. This enables
additional compile-time and run-time checks for several libc functions. To disable, specify either
-U_FORTIFY_SOURCE or -D_FORTIFY_SOURCE=0.

NOTE: In Debian 13 and Ubuntu 24.04 and later versions, -D_TIME_BITS=64 together with
-D_FILE_OFFSET_BITS=64 is set by default on the 32bit architectures armel, armhf, hppa, m68k, mips,
mipsel, powerpc and sh4.

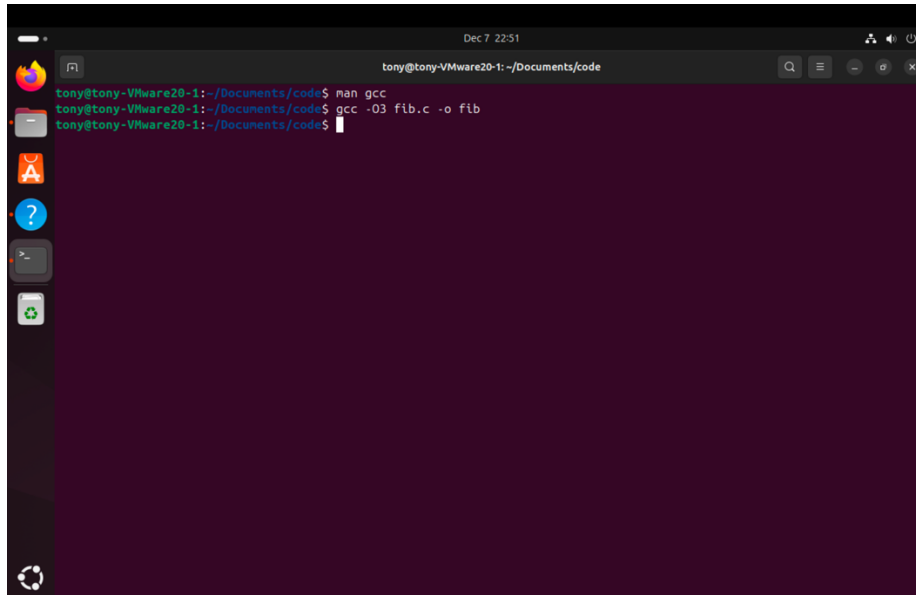
-O3 Optimize yet more. -O3 turns on all optimizations specified by -O2 and also turns on the following
optimization flags:

-fgcse-after-reload -fipa-cp-clone -floop-interchange -floop-unroll-and-jam -fpeel-loops
-fpredictive-commoning -fsplit-loops -fsplit-paths -ftree-loop-distribution -ftree-partial-pre
-funswitch-loops -fvect-cost-model=dynamic -fversion-loops-for-strides

-O0 Reduce compilation time and make debugging produce the expected results. This is the default.

Manual page gcc(1) line 7613 (press h for help or q to quit)
```

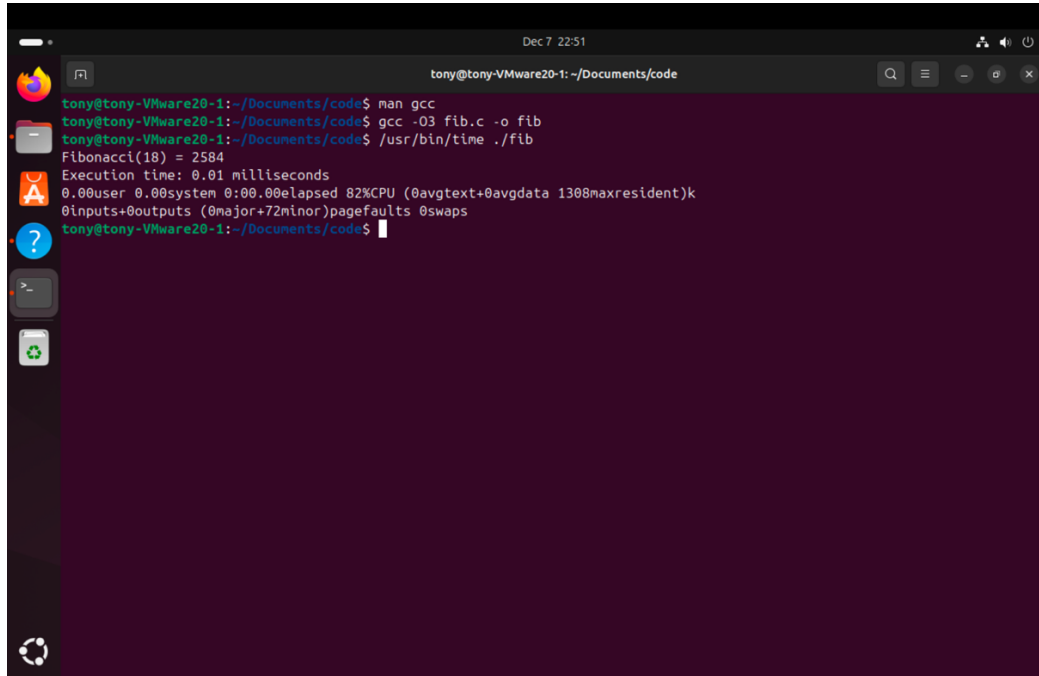
- b) Compile **fib.c** again with the optimization parameters



```
Dec 7 22:51
tony@tony-VMware20-1: ~/Documents/code

tony@tony-VMware20-1: ~/Documents/code$ man gcc
tony@tony-VMware20-1: ~/Documents/code$ gcc -O3 fib.c -o fib
tony@tony-VMware20-1: ~/Documents/code$
```

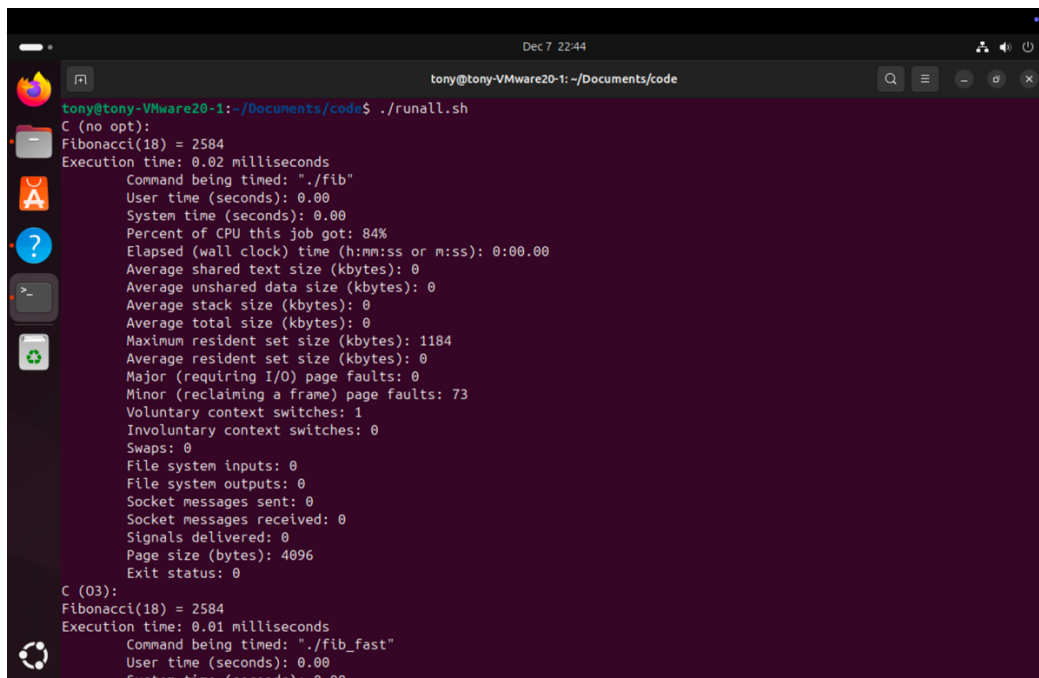
- c) Run the newly compiled program. Is it true that it now performs the calculation faster?



```
Dec 7 22:51
tony@tony-VMware20-1: ~/Documents/code
tony@tony-VMware20-1:~/Documents/code$ man gcc
tony@tony-VMware20-1:~/Documents/code$ gcc -O3 fib.c -o fib
tony@tony-VMware20-1:~/Documents/code$ /usr/bin/time ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
0.00user 0.00system 0:00.00elapsed 82%CPU (0avgtext+0avgdata 1308maxresident)k
0inputs+0outputs (0major+72minor)pagefaults 0swaps
tony@tony-VMware20-1:~/Documents/code$
```

It is sure faster (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.



```
Dec 7 22:44
tony@tony-VMware20-1: ~/Documents/code
tony@tony-VMware20-1:~/Documents/code$ ./runall.sh
C (no opt):
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
Command being timed: "./fib"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 84%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1184
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 73
Voluntary context switches: 1
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

C (O3):
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
Command being timed: "./fib_fast"
User time (seconds): 0.00
System time (seconds): 0.00
```

```
Dec 7 22:44
tony@tony-VMware20-1: ~/Documents/code

Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
Command being timed: "./fib_fast"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 85%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1304
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 72
Voluntary context switches: 1
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

Java:
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
Command being timed: "java Fibonacci"
User time (seconds): 0.05
System time (seconds): 0.01
Percent of CPU this job got: 108%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.05
Exit status: 0

Java:
Fibonacci(18) = 2584
Execution time: 0.27 milliseconds
Command being timed: "java Fibonacci"
User time (seconds): 0.05
System time (seconds): 0.01
Percent of CPU this job got: 108%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.05
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 42172
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 5415
Voluntary context switches: 224
Involuntary context switches: 12
Swaps: 0
File system inputs: 0
File system outputs: 64
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

Python:
Fibonacci(18) = 2584
Execution time: 0.21 milliseconds
Command being timed: "python3 fib.py"
User time (seconds): 0.00
System time (seconds): 0.00
```

The image shows two screenshots of a terminal window. The top screenshot shows the output of a Python script (fibonacci) and a shell command (time ./fib\_sh). The bottom screenshot shows the output of the same shell command (time ./fib\_sh) and a shell command (time ./fib\_sh).

```
Dec 7 22:44
tony@tony-VMware20-1: ~/Documents/code

Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

Python:
Fibonacci(18) = 2584
Execution time: 0.21 milliseconds
Command being timed: "python3 fib.py"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 100%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 8844
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 868
Voluntary context switches: 1
Involuntary context switches: 2
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

Bash:
/usr/bin/time: cannot run ./fib_sh: No such file or directory
Command exited with non-zero status 127
Command being timed: " ./fib_sh"
```

```
Dec 7 22:45
tony@tony-VMware20-1: ~/Documents/code

Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

Bash:
/usr/bin/time: cannot run ./fib_sh: No such file or directory
Command exited with non-zero status 127
Command being timed: " ./fib_sh"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 66%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 972
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 41
Voluntary context switches: 1
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 127

tony@tony-VMware20-1: ~/Documents/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.

The screenshot shows the OakSim ARM simulator interface. On the left, the assembly code is displayed with line numbers 1 through 12. The code includes instructions for moving registers, multiplying, subtracting, comparing, and branching. On the right, the Register Value table shows the state of registers R0 through R12. Below the registers, a memory dump shows the contents of memory addresses from 0x00010000 to 0x00010210. The memory dump shows a sequence of bytes that correspond to the decimal value 16 (0x10) repeated multiple times.

```
1 Main: //588963 Tony Jarwa
2 mov r0, #1
3 mov r1, #2
4 mov r2, #4
5 Loop:
6 mul r0, r0, r1
7 sub r2, r2, #1
8 cmp r2, #4
9 beq End
10 b Loop
11 End:
12 |
```

Register	Value
R0	10
R1	2
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0

Memory Dump (hex):

```
0x00010000: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010010: 01 20 42 E2 00 00 00 00 00 00 00 00 00 00 00 00
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
0x000100A0: 00 00 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 00 00
0x000100C0: 00 00 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 00 00
0x000100E0: 00 00 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 00 00
0x00010100: 00 00 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 00 00
0x00010120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000101A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000101B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000101C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000101D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000101E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000101F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

10 in Hex is 16 in Decimal.

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