

Template Week 4 – Software

Student number: 575798

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim ARM assembly debugger interface. On the left, the assembly code for calculating a factorial is displayed:

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

On the right, the register values are listed:

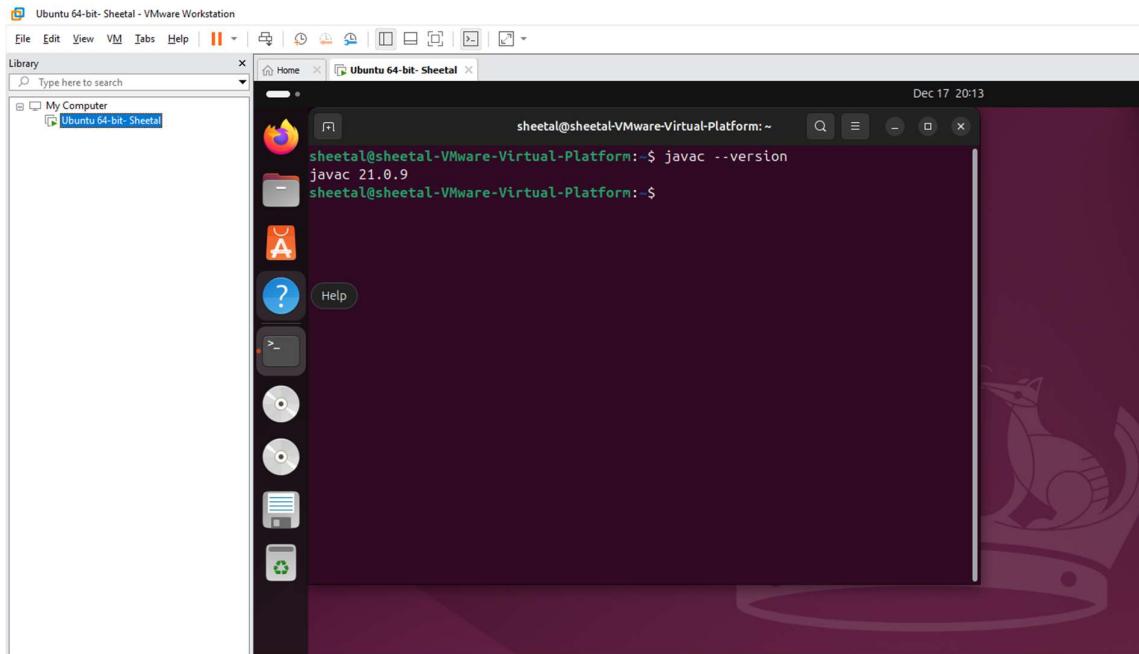
Register	Value
R0	0
R1	78
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
SP	10000

The memory dump area shows the state of memory at various addresses, with the first few bytes of memory containing the assembly code itself.

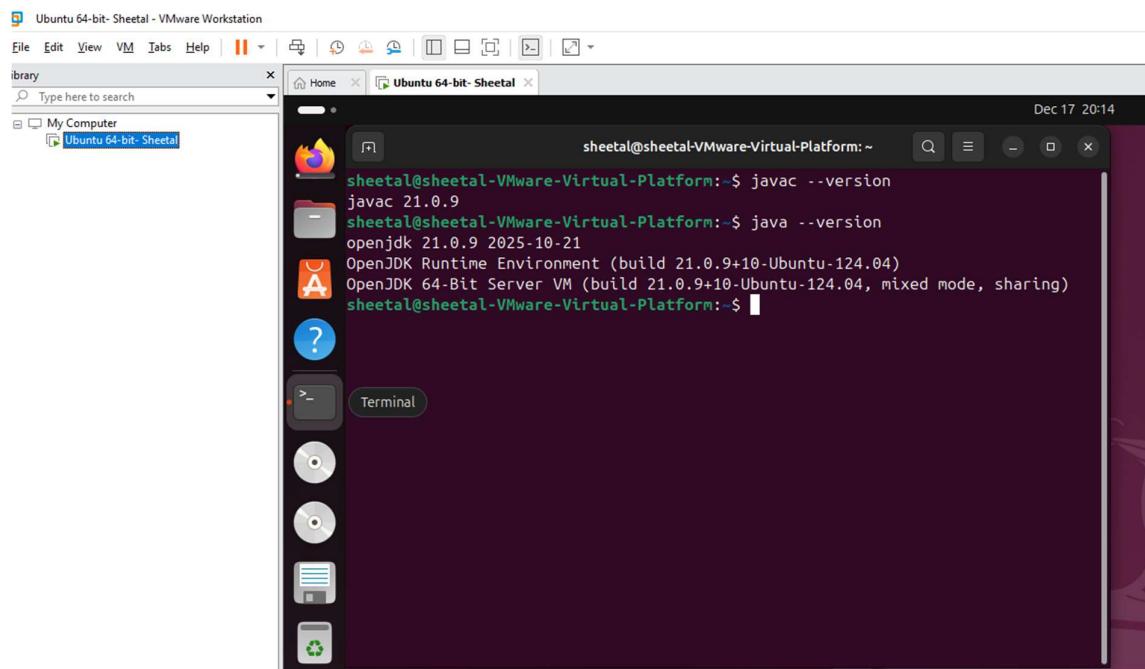
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac –version



java –version



gcc –version

```
sheetal@sheetal-VMware-Virtual-Platform: ~$ javac --version
javac 21.0.9
sheetal@sheetal-VMware-Virtual-Platform: ~$ 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)
sheetal@sheetal-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.

sheetal@sheetal-VMware-Virtual-Platform: ~$
```

python3 –version

```
sheetal@sheetal-VMware-Virtual-Platform: ~$ javac --version
javac 21.0.9
sheetal@sheetal-VMware-Virtual-Platform: ~$ 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)
sheetal@sheetal-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.

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

bash –version

```
sheetal@sheetal-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
sheetal@sheetal-VMware-Virtual-Platform:~$ 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)
sheetal@sheetal-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.

sheetal@sheetal-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
sheetal@sheetal-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>

sheetal@sheetal-VMware-Virtual-Platform:~$
```

Assignment 4.3: Compile

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

The C source file needs to be compiled. The java source file needs to be compiled. The python file and bash script does not need to be compiled.

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

The C compiler (gcc) compiles C code directly into native machine code, which is executed directly by the CPU.

Which source code files are compiled to byte code?

Java source code is compiled by java c into Java bytecode which runs on the Java Virtual Machine.

Which source code files are interpreted by an interpreter?

Python (.py) interpreted by the Python interpreter

Bash (.sh) interpreted by the Bash shell

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

C program is expected to be the fastest since it is compiled into native machine code. Also, because there is no virtual machine or interpreter overhead and it runs directly on the CPU.

How do I run a Java program?

Compile javac Fib.java

How do I run a Python program?

python3 fib.py

How do I run a C program?

gcc fib.c -o fib

How do I run a Bash script?

sudo chmod a+x fib.sh

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

For java there will be a bytecode file and for C there will be an executable file. For python and bash there will be no compiled file.

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?

```
sheetal@sheetal-VMware-Virtual-Platform:~$ gcc fib.c -o fib
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetal@sheetal-VMware-Virtual-Platform:~$ cd code
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
sheetal@sheetal-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib
javac Fibonacci.java
sheetal@sheetal-VMware-Virtual-Platform:~/code$ chmod a+x fib
sudo chmod a+x fib.sh
[sudo] password for sheetal:
sheetal@sheetal-VMware-Virtual-Platform:~/code$
```

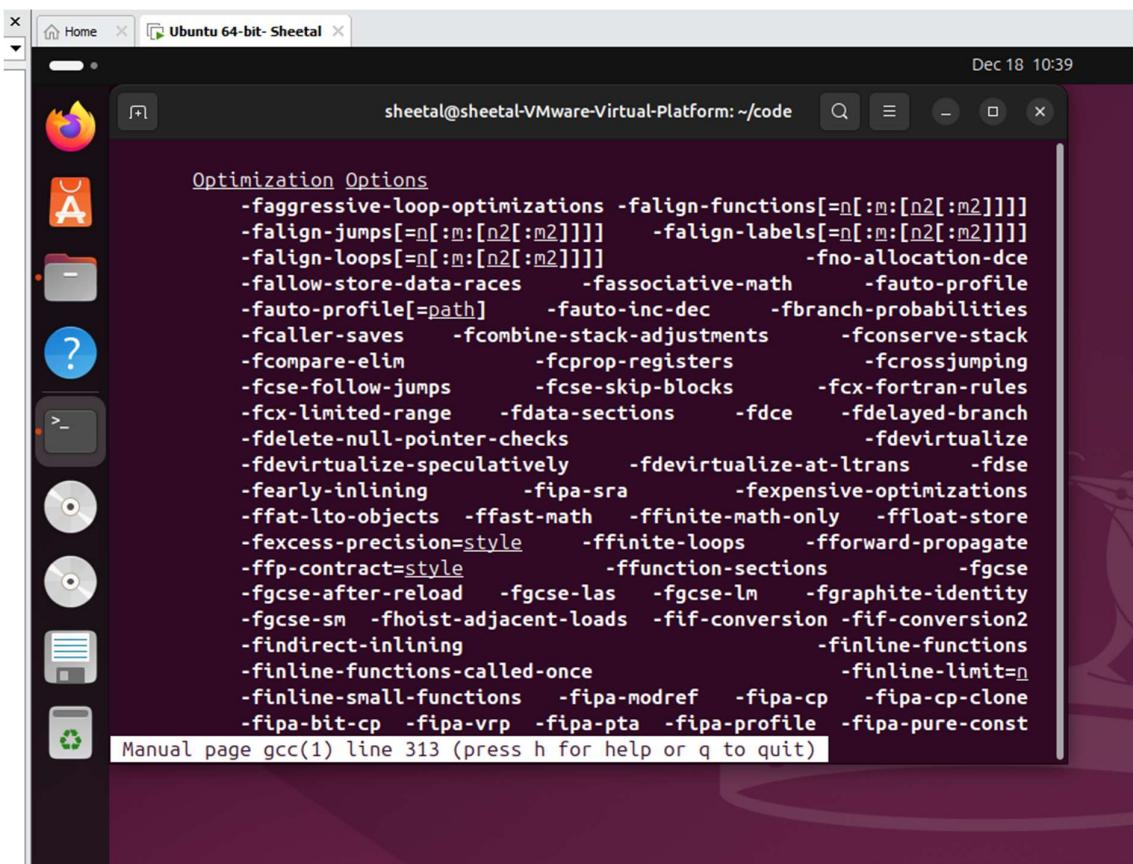
```
sheetal@sheetal-VMware-Virtual-Platform:~$ gcc fib.c -o fib
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetal@sheetal-VMware-Virtual-Platform:~$ cd code
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ls
fib.c Fibonacci.java fib.py fib.sh runall.sh
sheetal@sheetal-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib
javac Fibonacci.java
sheetal@sheetal-VMware-Virtual-Platform:~/code$ chmod a+x fib
sudo chmod a+x fib.sh
[sudo] password for sheetal:
sheetal@sheetal-VMware-Virtual-Platform:~/code$ ./fib
java Fibonacci
python3 fib.py
sudo ./fib.sh
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
Fibonacci(18) = 2584
Execution time: 0.19 milliseconds
Fibonacci(18) = 2584
Execution time: 0.24 milliseconds
Fibonacci(18) = 2584
Excution time 4758 milliseconds
sheetal@sheetal-VMware-Virtual-Platform:~/code$
```

C calculates the fastest.

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.

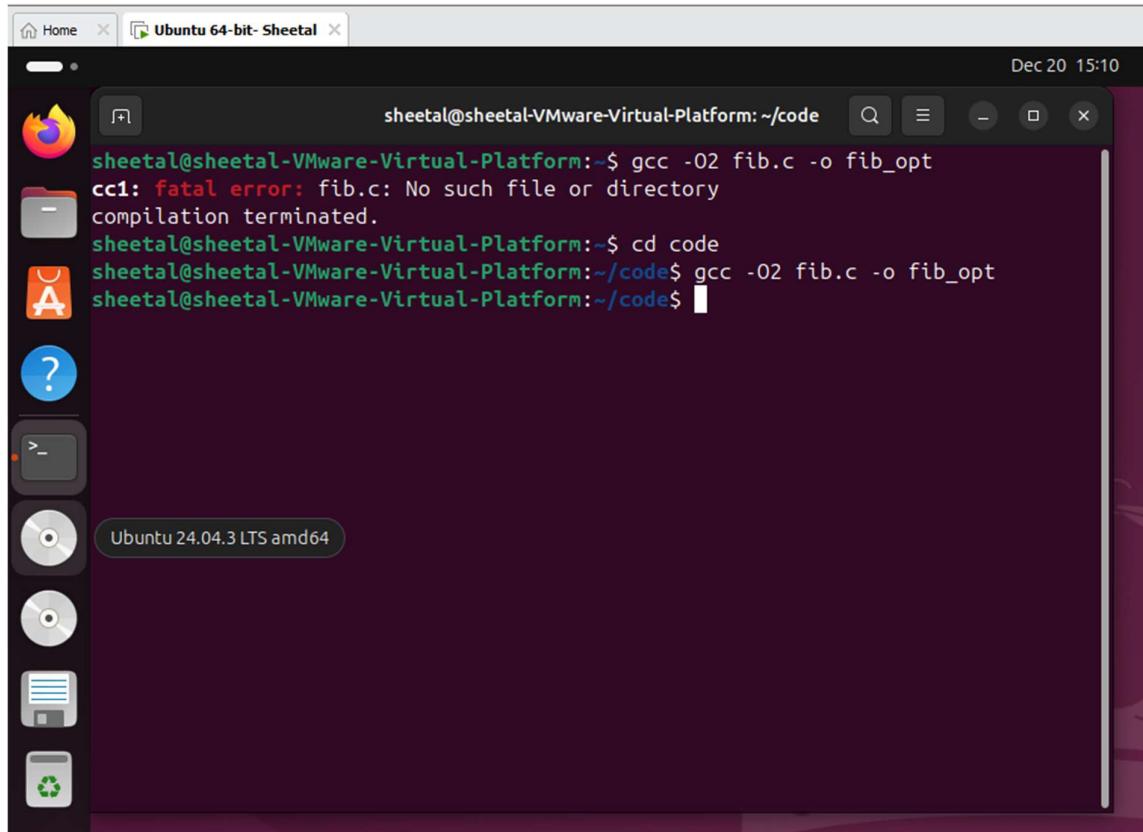


The screenshot shows a terminal window titled 'Ubuntu 64-bit- Sheetal'. The command 'gcc -fhelp' has been entered, resulting in a large list of optimization options. The options listed include:
-faggressive-loop-optimizations -falign-functions[=n[:m:[n2[:m2]]]]
-falign-jumps[=n[:m:[n2[:m2]]]] -falign-labels[=n[:m:[n2[:m2]]]]
-falign-loops[=n[:m:[n2[:m2]]]] -fno-allocation-dce
-fallow-store-data-races -fassociative-math -fauto-profile
-fauto-profile[=path] -fauto-inc-dec -fbranch-probabilities
-fcaller-saves -fcombine-stack-adjustments -fconserve-stack
-fcompare-elim -fcprop-registers -fcrossjumping
-fcse-follow-jumps -fcse-skip-blocks -fcx-fortran-rules
-fcx-limited-range -fdata-sections -fdce -fdelayed-branch
-fdelete-null-pointer-checks -fdevirtualize
-fdevirtualize-speculatively -fdevirtualize-at-ltrans -fdse
-fearly-inlining -fipa-sra -fexpensive-optimizations
-ffat-lto-objects -ffast-math -ffinite-math-only -ffloat-store
-fexcess-precision=style -ffinite-loops -fforward-propagate
-ffp-contract=style -ffunction-sections -fgcse
-fgcse-after-reload -fgcse-las -fgcse-lm -fgraphite-identity
-fgcse-sm -fhoist-adjacent-loads -fif-conversion -fif-conversion2
-findirect-inlining -finline-functions
-finline-functions-called-once -finline-limit=n
-finline-small-functions -fipa-modref -fipa-cp -fipa-cp-clone
-fipa-bit-cp -fipa-vrp -fipa-pta -fipa-profile -fipa-pure-const

Manual page gcc(1) line 313 (press h for help or q to quit)

I can see that gcc supports multiple optimisation flags. The recommended optimisation level is -O2 because there is a wider range of optimisations that can increase the speed.

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

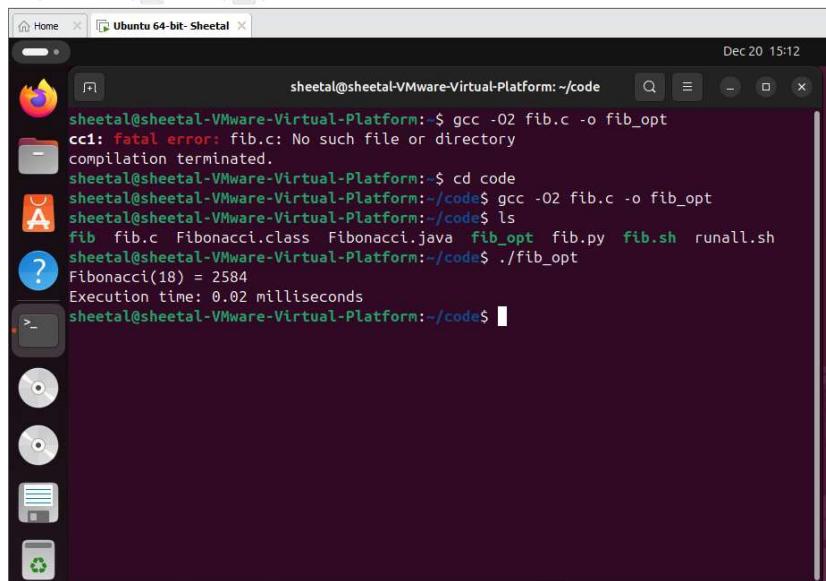


A screenshot of a Ubuntu 64-bit desktop environment. The terminal window shows the following command-line session:

```
sheetal@sheetsl-VMware-Virtual-Platform: ~/code
$ gcc -O2 fib.c -o fib_opt
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetal@sheetsl-VMware-Virtual-Platform: ~/code
$ cd code
sheetal@sheetsl-VMware-Virtual-Platform: ~/code
$ gcc -O2 fib.c -o fib_opt
sheetal@sheetsl-VMware-Virtual-Platform: ~/code
```

The desktop interface includes a dock with icons for Home, Dash, Applications, Help, and a terminal, along with a system tray icon for the date and time.

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



A screenshot of a Ubuntu 64-bit desktop environment. The terminal window shows the following command-line session, demonstrating a faster execution time for the optimized Fibonacci calculation:

```
sheetsl@sheetsl-VMware-Virtual-Platform: ~/code
$ gcc -O2 fib.c -o fib_opt
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
sheetsl@sheetsl-VMware-Virtual-Platform: ~/code
$ cd code
sheetsl@sheetsl-VMware-Virtual-Platform: ~/code
$ gcc -O2 fib.c -o fib_opt
sheetsl@sheetsl-VMware-Virtual-Platform: ~/code
$ ls
fib fib.c Fibonacci.class Fibonacci.java fib_opt fib.py fib.sh runall.sh
sheetsl@sheetsl-VMware-Virtual-Platform: ~/code
$ ./fib_opt
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
sheetsl@sheetsl-VMware-Virtual-Platform: ~/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.

```
Running C program:  
Fibonacci(19) = 4181  
Execution time: 0.08 milliseconds  
  
Running Java program:  
Fibonacci(19) = 4181  
Execution time: 0.31 milliseconds  
  
Running Python program:  
Fibonacci(19) = 4181  
Execution time: 0.39 milliseconds  
  
Running BASH Script  
Fibonacci(19) = 4181  
Excution time 8339 milliseconds  
  
sheetal@sheetal-VMware-Virtual-Platform:~/code$
```

As we can see C program performs the calculations the fastest.

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 QEMU debugger interface with the following details:

- Registers:**

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
SP	10000
- Memory Dump:** A large block of memory dump data starting at address 0x000010000, showing hex values for each byte.
- Stack Trace:** A stack trace showing the call history:


```

abort() at Error at jsStackTrace (https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:5:1882)
at Object.error (https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:5:1882) at Object.abort
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:29:7211) at _abort
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:5:20032) at Array.SB
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:5:20032) at invoke_IAM [as dynCall_iiii]
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:9:17068) at invoke_iii
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:5:284718) at WC
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:5:284718) at AC
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:19:99989) at Array.RC
(https://wankolo.github.io/QekSim/lib/unicorn-arm.min.js:19:99989) at Array.RC
      
```

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