

Template Week 4 – Software

Student number: 579185 – Nafsika Pagkali

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim interface. At the top, there's a navigation bar with back, forward, and search icons, and the URL 'wunkolo.github.io/OakSim/'. Below the URL is the OakSim logo. On the left, there's a control panel with buttons for 'Open', 'Run' (which is highlighted), '250', 'Step', and 'Reset'. In the center, the assembly code is displayed:

```
1 Main:  
2 mov r2,#5  
3 mov r0,#1  
4 Loop:  
5 sub r3,r2,r0  
6 sub r4,r3,r0  
7 sub r5,r4,r0  
8 mul r6,r2,r3  
9 mul r7,r4,r5  
10 mul r1,r6,r7  
11  
12 End:
```

To the right, there's a table titled 'Register Value' showing the current state of the registers:

Register	Value
R0	1
R1	78
R2	5
R3	4
R4	3
R5	2
R6	14
R7	6

Below the register table, memory dump values are shown:

0x00010000:	05	20	A0
0x00010010:	00	50	44
0x00010020:	00	00	00
0x00010030:	00	00	00
0x00010040:	00	00	00
0x00010050:	00	00	00

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

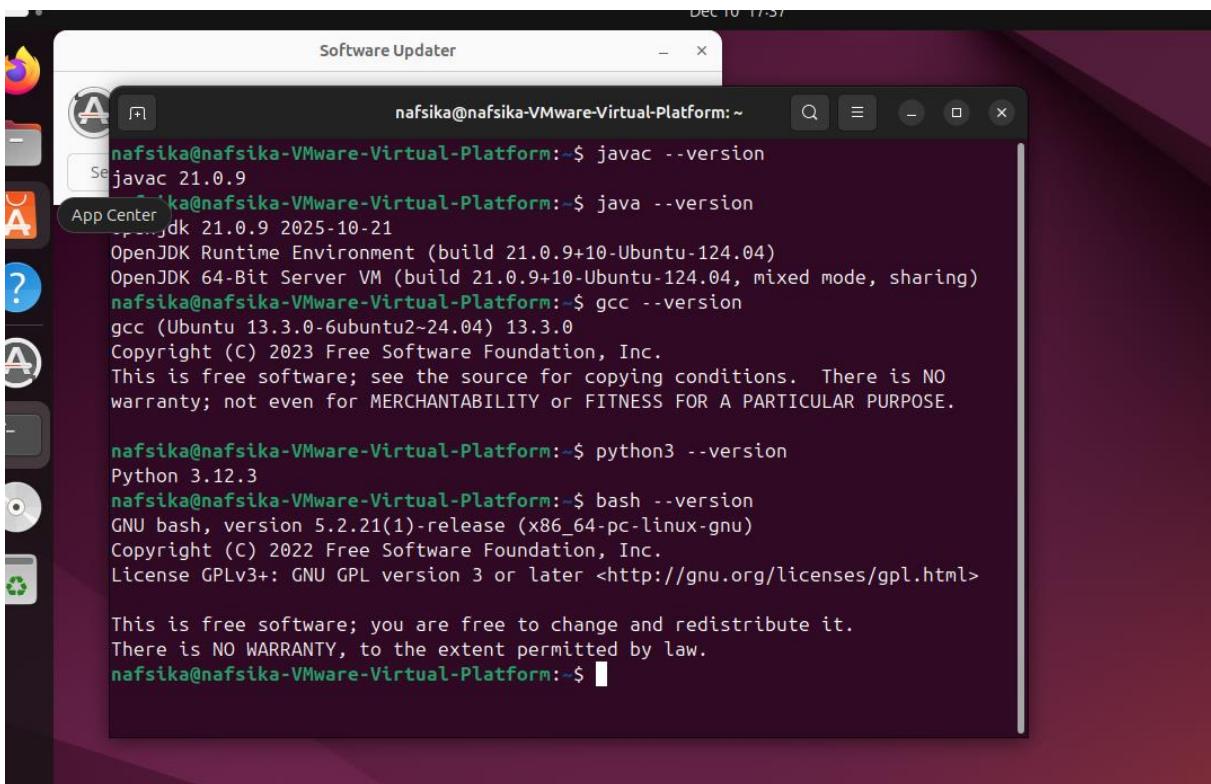
javac --version

java --version

gcc --version

python3 --version

bash --version



The image shows a screenshot of a Linux desktop environment, likely Ubuntu, with a dark theme. A terminal window titled "Software Updater" is open in the foreground, displaying command-line output. The terminal window has a title bar with the title "Software Updater" and a date/time stamp "Dec 10 17:57". The main area of the terminal shows the following command-line session:

```
nafsika@nafsika-Virtual-Platform: ~$ javac --version
javac 21.0.9
  ~nafsika@nafsika-Virtual-Platform: ~$ java --version
  jdk 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)
nafsika@nafsika-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.

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

Assignment 4.3: Compile

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

Java and C

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

C

Which source code files are compiled to byte code?

Java

Which source code files are interpreted by an interpreter?

Python and Bash

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

C because machine code is executed by the processor

How do I run a Java program?

`javac Fibonacci.java` (to compile)

`java Fibonacci`

How do I run a Python program?

`python3 fib.py`

How do I run a C program?

`gcc fib.c -o fib` (to compile)

`./fib`

How do I run a Bash script?

Bash `fib.sh`

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

Java and C will create new files but python and bash no

Java : will create Fibonacci class

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?
C and after that Java

```

nafsika@nafsika-Virtual-Platform:~/Downloads$ ls -l
total 20
drw-rw-r-- 1 nafsika nafsika 831 Jun  9  2023 fib.c
drw-rw-r-- 1 nafsika nafsika 839 Jun  9  2023 Fibonacci.java
drw-rw-r-- 1 nafsika nafsika 516 Jun  9  2023 fib.py
drw-rw-r-- 1 nafsika nafsika 668 Jun  9  2023 fib.sh
drw-rw-r-- 1 nafsika nafsika 249 Jun  9  2023 runall.sh
+nafsika@nafsika-Virtual-Platform:~/Downloads/code$ javac Fibonacci.java
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.29 milliseconds
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ $

nafsika@nafsika-Virtual-Platform:~/Downloads$ gcc fib.c -o fib
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
nafsika@nafsika-Virtual-Platform:~/Downloads$ cd code
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ gcc fib.c -o fib
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.07 milliseconds
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ $

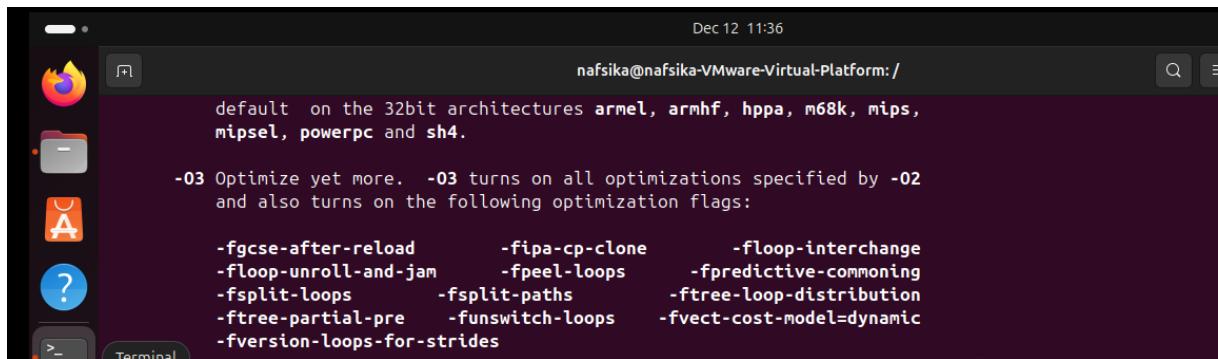
$ ./fib
Fibonacci(18) = 2584
Execution time 6107 milliseconds
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ chmod +x fib.sh
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 6010 milliseconds
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ $

$ python3 fib.py
Execution time 6010 milliseconds
nafsika@nafsika-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.68 milliseconds
nafsika@nafsika-Virtual-Platform:~/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. `gcc -O3`

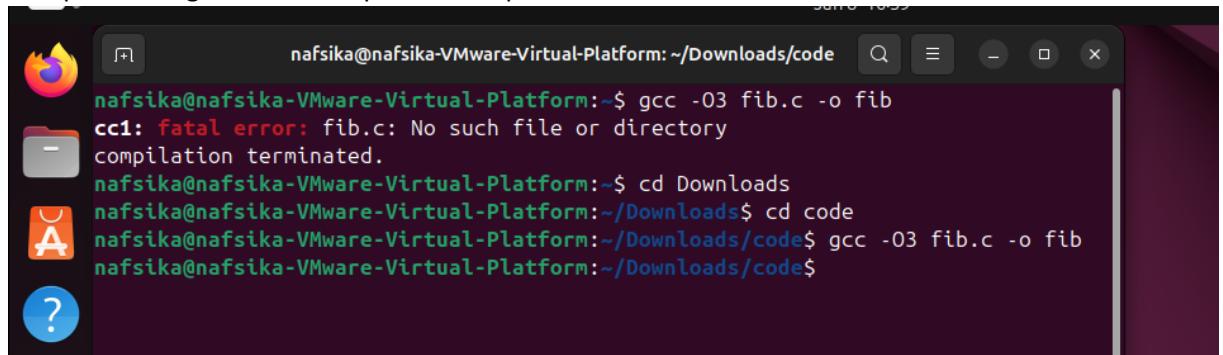


```
Dec 12 11:36
nafsiha@nafsiha-Virtual-Platform:~/Downloads$ gcc -O3
default on the 32bit architectures armel, armhf, hppa, m68k, mips,
mipsel, powerpc and sh4.

-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   -fvectorize-cost-model=dynamic
-fversion-loops-for-strides
```

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



```
nafsiha@nafsiha-Virtual-Platform:~/Downloads$ gcc -O3 fib.c -o fib
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
nafsiha@nafsiha-Virtual-Platform:~/Downloads$ cd Downloads
nafsiha@nafsiha-Virtual-Platform:~/Downloads$ cd code
nafsiha@nafsiha-Virtual-Platform:~/Downloads/code$ gcc -O3 fib.c -o fib
nafsiha@nafsiha-Virtual-Platform:~/Downloads/code$
```

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

```
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 fib.c -o fib  
cc1: fatal error: fib.c: No such file or directory  
compilation terminated.  
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads$ cd Downloads  
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads$ cd code  
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads/code$ gcc -O3 fib.c -o fib  
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads/code$ ./fib  
Fibonacci(18) = 2584  
Execution time: 0.02 milliseconds  
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads/code$
```

d)

- e) 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: 0.01 milliseconds  
  
Running Java program:  
Fibonacci(19) = 4181  
Execution time: 0.46 milliseconds  
  
Running Python program:  
Fibonacci(19) = 4181  
Execution time: 0.72 milliseconds  
  
Running BASH Script  
Fibonacci(19) = 4181  
Execution time 12203 milliseconds  
  
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads/code$  
nafsi@nafsi-VMware-Virtual-Platform:~/Downloads/code$ S
```

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.

ain:

```
mov r0, #1    result = 1
mov r1, #2    base = 2
mov r2, #4    exponent = 4
```

Loop:

```
cmp r2, #0
beq End
mul r0, r0, r1
sub r2, r2, #1
b Loop
```

End:

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

Screenshot of the completed code here.

The screenshot shows the OakSim assembly debugger interface. The code editor on the left contains the assembly code with line numbers 1 through 14. The code includes instructions to move constants into registers R0, R1, and R2, and then enter a loop where it multiplies R0 by R1, decrements R2, and loops back until R2 is zero. The register window on the right shows all eight registers (R0-R7) initialized to 0. The memory dump window at the bottom shows the memory starting at address 0x000010000, containing the assembled binary code.

Register	Value
R0	10
R1	2
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0

Address	Value
0x000010000	01 00 A0 E3 02 10 A0 E3 04 20 A0 E3 00
0x000010010	02 00 00 0A 90 01 00 E0 01 20 42 E2 FA
0x000010020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010090	00 00 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](#)