

Template Week 4 – Software

Student number:592513

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows the OakSim software interface. At the top, there are buttons for 'Open', 'Run' (which is highlighted), 'Step', and 'Reset'. To the right of these are controls for setting the clock speed to 250 Hz and a 'Run' button with a play/pause icon. Below this is a text area containing ARM assembly code. The code starts with a main loop that initializes registers R2 and R1, then enters a loop where it multiplies R1 by R2, subtracts 1 from R2, and compares R2 with 1. If R2 is not 1, it loops back. Once R2 reaches 1, it exits the loop and goes to the end. The assembly code is as follows:

```
1 Main:
2   mov r2, #5
3   mov r1, #1
4
5   Loop:
6
7     mul r1, r2, r1
8     sub r2, r2, #1
9     cmp r2, #1
10    beq Exit
11    b Loop
12
13 Exit:
14
15 End:
```

Below the assembly code is a table titled 'Registers' showing the current values of all ARM registers (R0-R9). The values are as follows:

Register	Value
R0	0
R1	78
R2	1
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0

At the bottom of the screen, memory dump sections are shown for addresses 0x00010000 through 0x00010180. The memory contains various assembly instructions and their corresponding binary representations.

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

java --version

gcc --version

python3 --version

bash --version

```
david@david-VMware-Virtual-Platform:~$ javac --version
javac: command not found
david@david-VMware-Virtual-Platform:~$ java --version
Command 'java' not found, but can be installed with:
  sudo apt install openjdk-17-jdk-headless # version 17.0.16+8-us1-0ubuntu1-24.04.1, or
  sudo apt install openjdk-21-jdk-headless # version 21.0.8+9-us1-0ubuntu1-24.04.1
  sudo apt install default-jdk          # version 21.0.17-75
  sudo apt install ecj                  # version 3.32.0+eclipse4.26-2
  sudo apt install openjdk-19-jdk-headless # version 19.0.2+7-4
  sudo apt install openjdk-20-jdk-headless # version 20.0.2+9-1
  sudo apt install openjdk-22-jdk-headless # version 22-22ea-1
  sudo apt install openjdk-11-jdk-headless # version 11.0.28+6-1ubuntu1-24.04.1
  sudo apt install openjdk-8-jdk-headless # version 8u462-ga-us1-0ubuntu2-24.04.2
david@david-VMware-Virtual-Platform:~$ gcc --version
Command 'gcc' not found, but can be installed with:
  sudo apt install gcc
david@david-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
david@david-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.
david@david-VMware-Virtual-Platform:~$ java --version
Command 'java' not found, but can be installed with:
  sudo apt install openjdk-17-jre-headless # version 17.0.16+8-us1-0ubuntu1-24.04.1, or
  sudo apt install openjdk-21-jre-headless # version 21.0.8+9-us1-0ubuntu1-24.04.1
  sudo apt install default-jre          # version 21.0.17-75
  sudo apt install openjdk-19-jre-headless # version 19.0.2+7-4
  sudo apt install openjdk-20-jre-headless # version 20.0.2+9-1
  sudo apt install openjdk-22-jre-headless # version 22-22ea-1
  sudo apt install openjdk-11-jre-headless # version 11.0.28+6-1ubuntu1-24.04.1
  sudo apt install openjdk-8-jre-headless # version 8u462-ga-us1-0ubuntu2-24.04.2
david@david-VMware-Virtual-Platform:~$
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Assignment 4.3: Compile

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

Java

C

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

Bash

python

Which source code files are compiled to byte code?

java

Which source code files are interpreted by an interpreter?

Python

bash

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

C

How do I run a Java program?

java MyProgram

How do I run a Python program?

python3 script.py

How do I run a C program?

./program

How do I run a Bash script?

./script.sh

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

The screenshot shows a terminal window with a dark background. On the left side, there is a vertical toolbar with several icons: a Firefox icon, a folder icon, a minus sign icon, a question mark icon, a greater than or equal to icon, a CD/DVD icon, and a recycle bin icon. The main area of the terminal displays the following text:

```
Running C program:  
Fibonacci(19) = 4181  
Execution time: 0.04 milliseconds  
  
Running Java program:  
Fibonacci(19) = 4181  
Execution time: 0.57 milliseconds  
  
Running Python program:  
Fibonacci(19) = 4181  
Execution time: 1.20 milliseconds  
  
Running BASH Script  
Fibonacci(19) = 4181  
Excution time 21852 milliseconds
```

At the bottom of the terminal window, the prompt is visible: `david@david-VMware-Virtual-Platform:~/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.
-O0
-O1
- Compile **fib.c** again with the optimization parameters
- Run the newly compiled program. Is it true that it now performs the calculation faster?

```
Running Python program:  
Fibonacci(19) = 4181  
Execution time: 1.20 milliseconds
```

```
Running BASH Script  
Fibonacci(19) = 4181  
Excution time 21852 milliseconds
```

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

It is a lot faster than before

- 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.

The screenshot shows a Linux desktop environment with several windows open:

- Ubuntu 64-bit**: Shows the output of a C program: "Running C program: Fibonacci(19) = 4181 Execution time: 0.01 milliseconds".
- Debian 12.x 64-bit**: Shows the output of a Java program: "Running Java program: Fibonacci(19) = 4181 Execution time: 0.51 milliseconds".
- My Computer**: Shows the output of a Python program: "Running Python program: Fibonacci(19) = 4181 Execution time: 0.97 milliseconds".
- Windows 11 x64**: Shows the output of a BASH Script: "Running BASH Script Fibonacci(19) = 4181 Excution time 14406 milliseconds".

In the bottom right corner of the desktop, there is a terminal window showing the command prompt: `david@david-VMware-Virtual-Platform:~/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 assembly debugger interface. On the left, the assembly code is displayed:

```
1 Main:  
2 mov r1, #2  
3 mov r2, #4  
4 mov r0, #1  
5  
6  
7 Loop:  
8  
9     mul r0, r0, r1  
10    sub r2, r2, #1  
11    cmp r2, #0  
12    beq Exit  
13    b Loop  
14  
15 Exit:  
16  
17 End:
```

On the right, the register values are shown:

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

The memory dump area shows the state of memory starting at address 0x00001000. The first few bytes are 02 10 40 E3 04 20 A0 E3 01 00 A0 E3 90 01 00 E0, followed by several FF FF EA B R entries, indicating memory corruption or undefined behavior.

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