

Template Week 4 – Software

Student number:

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows an ARM assembly simulator interface. On the left, the assembly code is displayed with line numbers 1 through 13. The code calculates the factorial of 5 using registers R1 and R2. The 'Run' button is highlighted, and the counter '250' is shown. On the right, a 'Register Value' table lists registers R0 through R10. Below the registers, a memory dump shows hexadecimal values for addresses from 0x00010000 to 0x000101E0. The dump includes some non-zero values at the beginning, such as 05 20 A0 E3 at 0x00010000.

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

Register	Value
R0	0
R1	78
R2	1
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0

Memory dump (hexadecimal values):

```
0x00010000: 05 20 A0 E3 01 10 A0 E3 91 02 01 E0 01 20 42 E2 ... B
0x00010010: 01 00 52 E3 00 00 00 00 00 00 00 00 00 00 00 00 ... 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
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
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

javac --version

```
russell-verver@russell-verver-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
russell-verver@russell-verver-VMware-Virtual-Platform:~$
```

java --version

```
russell-verver@russell-verver-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)
russell-verver@russell-verver-VMware-Virtual-Platform:~$
```

gcc --version

```
russell-verver@russell-verver-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.

russell-verver@russell-verver-VMware-Virtual-Platform:~$
```

python3 --version

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

bash --version

```
russell-verver@russell-verver-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.
russell-verver@russell-verver-VMware-Virtual-Platform:~$
```

Assignment 4.3: Compile

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

. Fib.c

. Fibonacci.java

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?

.Fip.py

.fib.sh

.runall.sh

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

. fib.c

How do I run a Java program?

Javac Fibonacci.java

Compileren → How do I run a Python program?

Uitvoeren → java Fibonacci

How do I run a C program?

Compileren → gcc fib.c -o fib

Uitvoeren → ./fib

How do I run a Bash script?

Eerst → `chmod +x fib.sh`

Uitvoeren → `./fib.sh`

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

`.C` → ja er word een uitvoerbaar bestand aan gemaakt

`.java` → ja er word een `.class` bestand aangemaakt

`.py` → deze code word niet gecompileerd

`.sh` → deze code word ook niet gecompileerd

Take relevant screenshots of the following commands:

- Compile the source files where necessary

Java

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ gcc fib.c -o fib
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ls
fib fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$
```

C

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ javac Fibonacci.java
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ls
fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
```

- Make them executable

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.16 milliseconds
```

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
```

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.22 milliseconds
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$
```

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 4053 milliseconds
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$
```

• Wich (compiled) source code file performs the calculation the fastest?

Dat was de code van c deze deed er maar 0.01 milliseconde over terwijl bash er 4 seconde over deed.

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.

-01

-02

-03

Hoe hoger het getal aangeeft hoe meer optimalisaties de compiler uitvoert.

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

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ gcc -o3 fib.c -o fib_3
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ls
fib fib_3 fib.c Fibonacci.class Fibonacci.java fib.py fib.sh runall.sh
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ./fib_3
```

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

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ ./fib_3
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
russell-verver@russell-verver-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.

```
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ time ./fib_3 javac Fibonacci.java time java Fibonacci time python3 fib.py time bash fib.sh
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.002s
user    0m0.000s
sys     0m0.002s
russell-verver@russell-verver-VMware-Virtual-Platform:~/code$ chmod +x runall.sh

Running C program:
Fibonacci(19) = 4181
Execution time: 0.02 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.21 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.38 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Execution time 6770 milliseconds

russell-verver@russell-verver-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 an ARM assembly simulator interface. On the left, the assembly code is displayed with line numbers 1 through 14. The code implements a loop to calculate 2^4. On the right, a table shows the current values of registers R0 through R9. Below the register table, a memory dump shows the first four words of memory, with the first word containing the decimal value 16 (0x00000010).

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

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

Memory dump (hex):

```
0x00010000: 02 10 A0 E3 04 20 A0 E3 01 00 A0 E3 00 00 52 E3
0x00010010: 02 00 00 0A 90 01 00 E0 01 20 42 E2 FA FF FF EA
0x00010020: 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
0x00010040: 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](#)