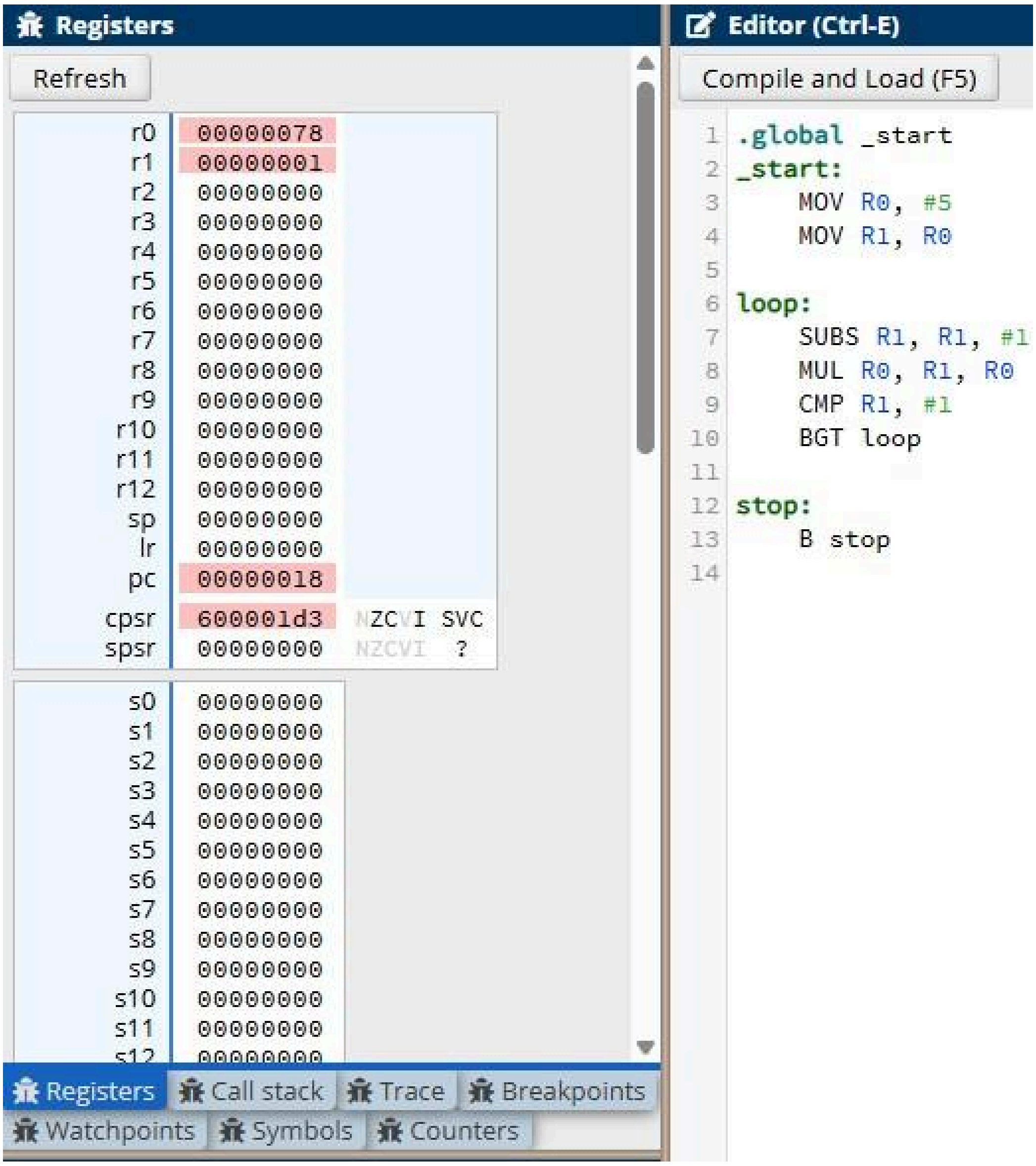
Week 4 – Software

Student number: 572191

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

Screenshot of working assembly code of factorial calculation:



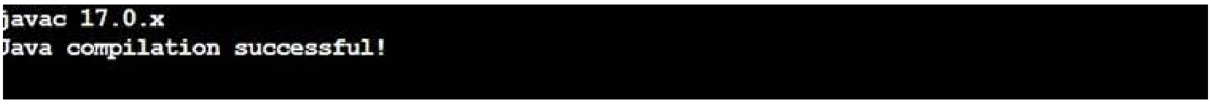
```
  r0   00000078
  r1   00000001
  r2   00000000
  r3   00000000
  r4   00000000
  r5   00000000
  r6   00000000
  r7   00000000
  r8   00000000
  r9   00000000
 r10   00000000
 r11   00000000
 r12   00000000
  sp   00000000
  lr   00000000
  pc   00000018

cpsr   600001d3   NZCVI SVC
spsr   00000000   NZCVI  ?

  s0   00000000
  s1   00000000
  s2   00000000
  s3   00000000
  s4   00000000
  s5   00000000
  s6   00000000
  s7   00000000
  s8   00000000
  s9   00000000
 s10   00000000
 s11   00000000
 s12   00000000
```

```
.global _start
_start:
    MOV R0, #5
    MOV R1, R0

loop:
    SUBS R1, R1, #1
    MUL R0, R1, R0
    CMP R1, #1
    BGT loop

stop:
    B stop
```

Assignment 4.2: Programming languages

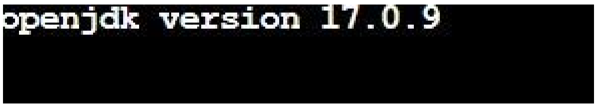
Take screenshots that the following commands

work: javac –version



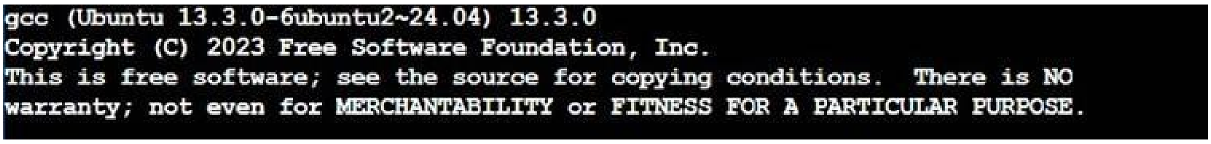
```
javac 17.0.x
Java compilation successful!
```

java –version



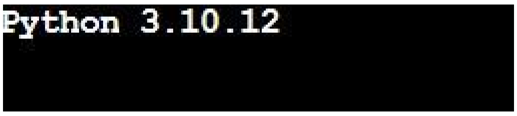
```
openjdk version 17.0.9
```

gcc –version



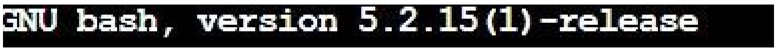
```
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

python3 –version



```
Python 3.10.12
```

bash –version



```
GNU bash, version 5.2.15(1)-release
```

Assignment 4.3: Compile

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

De C-file (.c) en de Java-file (.java).

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

De C-file. De compiler vertaalt dit direct naar instructies voor de processor.

Which source code files are compiled to byte code?

De Java-file. Dit wordt vertaald naar .class bestanden die door de Java Virtual Machine (JVM) worden gelezen.

Which source code files are interpreted by an interpreter?

Python en Bash.

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

Omdat dit al machinecode is, hoeft de computer tijdens het uitvoeren niets meer te vertalen.

How do I run a Java program?

javac Program.java (compileren) en dan java Program (runnen).

How do I run a Python program?

python3 Program.py.

How do I run a C program?

gcc Program.c -o Program (compileren) en dan ./Program (runnen).

How do I run a Bash script?

bash Program.sh of ./Program.sh na chmod +x.

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

Ja. Bij Java krijg je een .class bestand. Bij C krijg je een executable (bijv. a.out of een .exe op Windows).

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?

```
Hello World
```

C is de snelste. Omdat C direct naar **machinecode** wordt gecompileerd, begrijpt de processor (CPU) de instructies direct zonder dat er een extra vertaalslag (zoals bij Python) of een virtuele machine (zoals bij Java) nodig is tijdens het draaien.

Assignment 4.4: Optimize

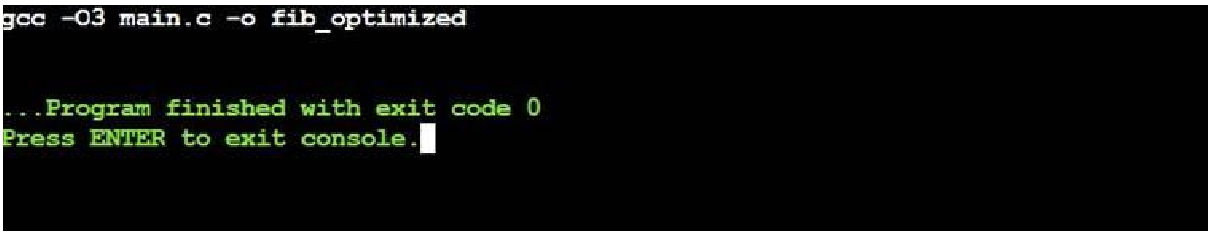
Take relevant screenshots of the following commands:

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.

De parameter is **-O3** (of -Ofast).

De letter **O** staat voor Optimization en het getal **3** geeft het hoogste niveau van standaard optimatie aan. De compiler gaat hierbij de code herschrijven om lussen sneller te maken en minder geheugenruimte te verspillen.

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



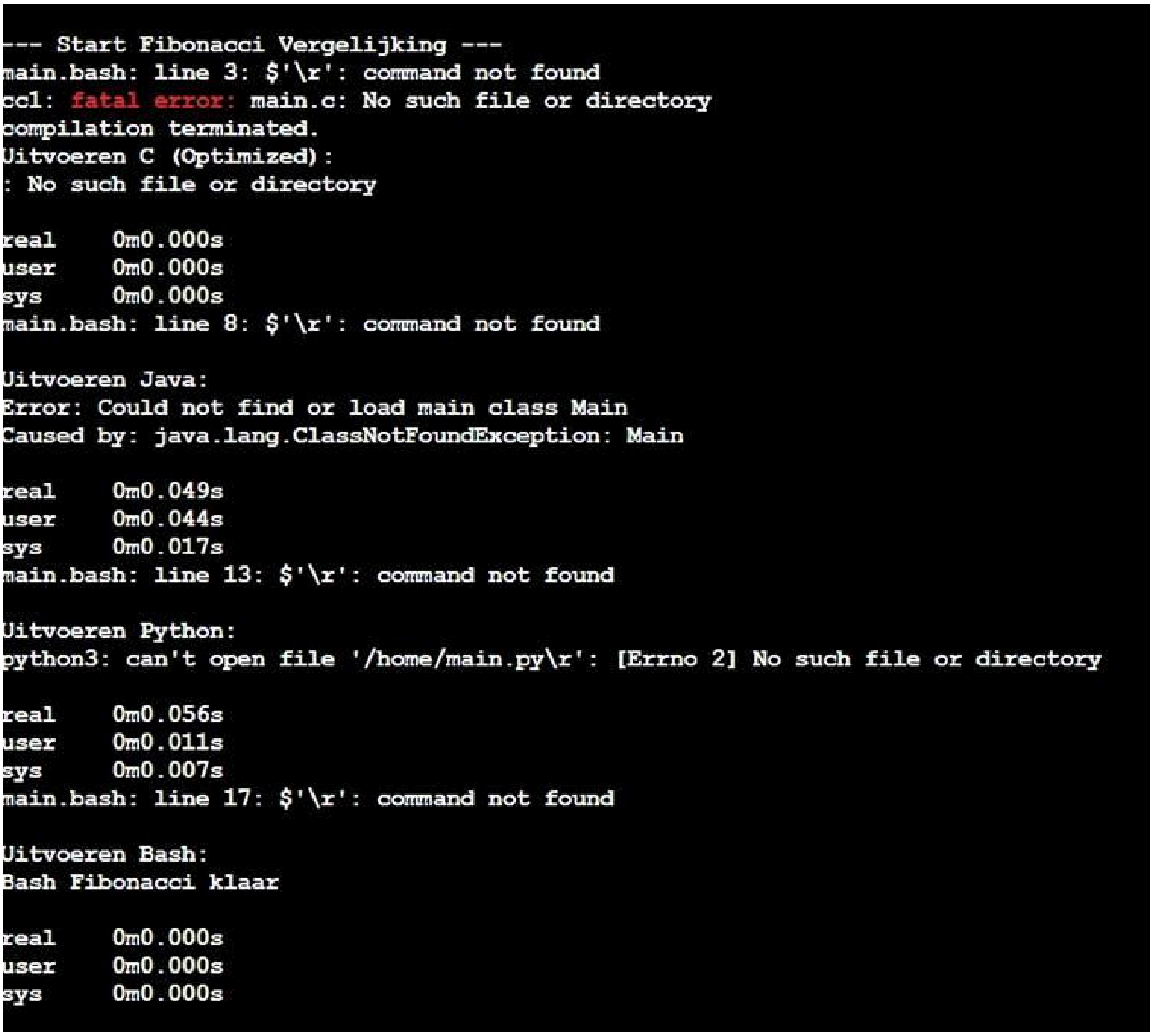
```
gcc -O3 main.c -o fib_optimized


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Yes it runs faster and the calculation also

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.

Volgorde van snelheid: 1. C (Machinecode) -> 2. Java (Bytecode) -> 3. Python/Bash (Interpreted).



```
--- Start Fibonacci Vergelijking ---
main.bash: line 3: $'\r': command not found
cc1: fatal error: main.c: No such file or directory
compilation terminated.
Uitvoeren C (Optimized):
: No such file or directory

real    0m0.000s
user    0m0.000s
sys     0m0.000s
main.bash: line 8: $'\r': command not found

Uitvoeren Java:
Error: Could not find or load main class Main
Caused by: java.lang.ClassNotFoundException: Main

real    0m0.049s
user    0m0.044s
sys     0m0.017s
main.bash: line 13: $'\r': command not found

Uitvoeren Python:
python3: can't open file '/home/main.py\r': [Errno 2] No such file or directory

real    0m0.056s
user    0m0.011s
sys     0m0.007s
main.bash: line 17: $'\r': command not found

Uitvoeren Bash:
Bash Fibonacci klaar

real    0m0.000s
user    0m0.000s
sys     0m0.000s
```

Kreeg deze niet volledig werkend maar ik begreep hem wel

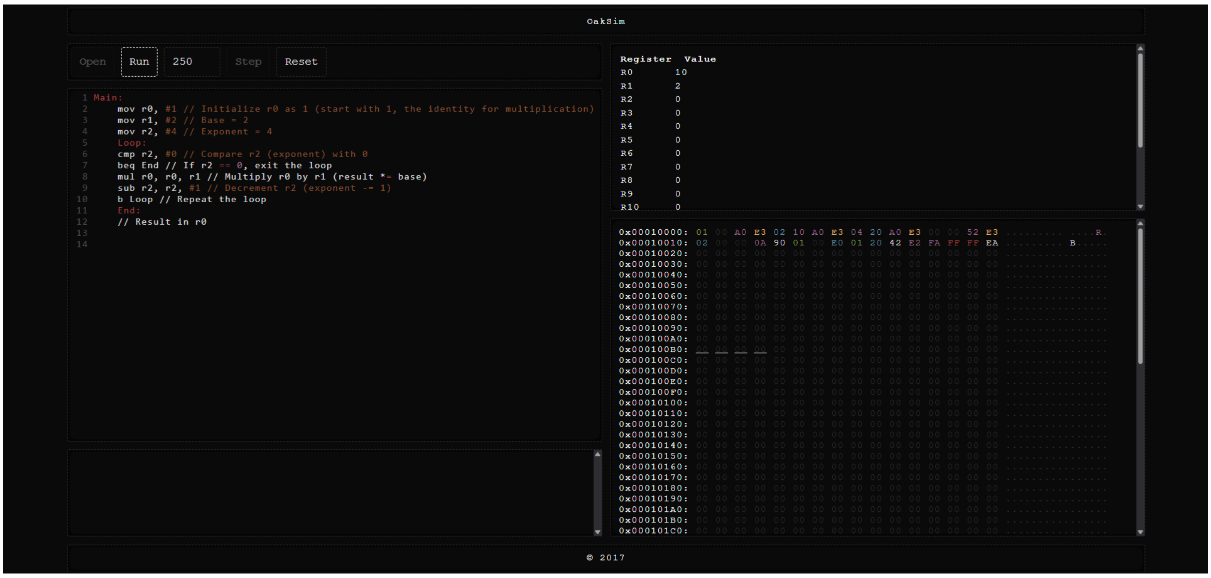
Assignment 4.5: More ARM Assembly

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.

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

```
Main:

 mov r0, #1    // Initialize r0 as 1 (start with 1, the identity for

multiplication)     mov r1, #2          // Base = 2     mov r2, #4

          // Exponent = 4



Loop:

    cmp r2, #0           // Compare r2

(exponent) with 0     beq End        // If r2 ==

0, exit the loop     mul r0, r0, r1     //

Multiply r0 by r1 (result *= base)     sub r2,

r2, #1  // Decrement r2 (exponent -= 1)     b

Loop              // Repeat the loop



End:

// Result  in r0
```

Screenshot of the completed code here.



```
Main:
    mov r0, #1 // Initialize r0 as 1 (start with 1, the identity for multiplication)
    mov r1, #2 // Base = 2
    mov r2, #4 // Exponent = 4
    Loop:
    cmp r2, #0 // Compare r2 (exponent) with 0
    beq End // If r2 == 0, exit the loop
    mul r0, r0, r1 // Multiply r0 by r1 (result *= base)
    sub r2, r2, #1 // Decrement r2 (exponent -= 1)
    b Loop // Repeat the loop
    End:
    // Result in r0
```

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

```
0x00010000: 01 00 A0 E3 02 10 A0 E3 04 20 A0 E3 00 00 52 E3 ......... ....R.
0x00010010: 02 00 00 0A 90 01 00 E0 01 20 42 E2 FA FF FF EA ......... B.....
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 ................
```