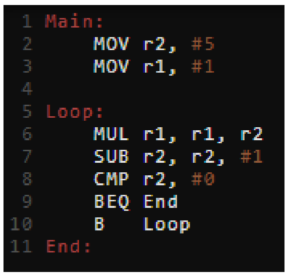
Template Week 4 – Software

Student number: 588421

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
    MOV r2, #5
    MOV r1, #1

Loop:
    MUL r1, r1, r2
    SUB r2, r2, #1
    CMP r2, #0
    BEQ End
    B   Loop
End:
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

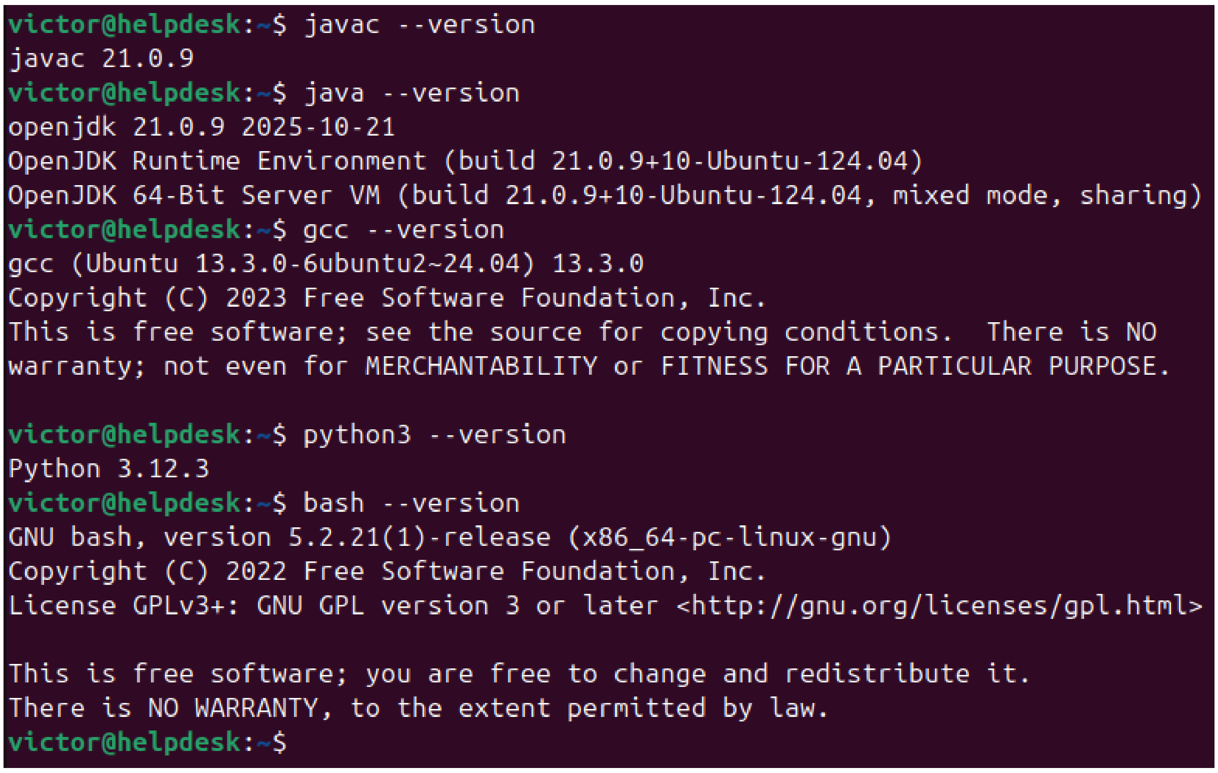
javac --version

java --version

gcc --version

python3 --version

bash --version



```
victor@helpdesk:~$ javac --version
javac 21.0.9
victor@helpdesk:~$ 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)
victor@helpdesk:~$ 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.

victor@helpdesk:~$ python3 --version
Python 3.12.3
victor@helpdesk:~$ 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.
victor@helpdesk:~$
```

Assignment 4.3: Compile

Which of the above files need to be compiled before you can run them?
Fibonacci.java moet eerst gecompileerd worden naar bytecode voordat je hem kan uitvoeren met de Java Virtual Machine.

fib.c moet gecompileerd worden naar machinecode

fib.py hoeft niet gecompileerd te worden de interpreter leest de code en voert die regel voor regel uit.

fib.sh wordt door /bin/bash geinterpreteerd en hoeft niet gecompileerd te worden. #!/usr/bin/env bash vertelt welk programma het script moet interpreteren.

Which source code files are compiled into machine code and then directly executable by a processor?
C wordt naar machine code gecompileerd en daarna door de CPU uitgevoerd

Which source code files are compiled to byte code?
Java compileerd naar bytecode

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?
C is het snelste omdat het native machine code is en de CPU de instructies direct uitvoerd
Daarna Java die compileet naar bytecode en draait op de JVM
Python wordt geinterpreteerd en de interpreter voert het vervolgens uit.
Bash is het traagst want de Shell interpreter textregels en start veel processen en commando's

How do I run a Java program?
javac Fibonacci.java

How do I run a Python program?
python3 fib.py

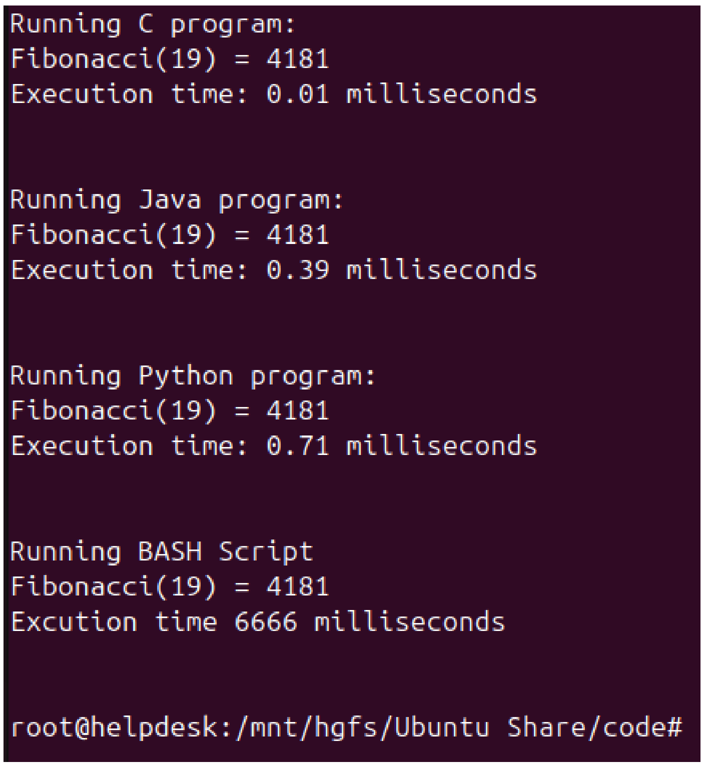
How do I run a C program?
gcc fib.c

How do I run a Bash script?
chmod -x fib.sh
fib.sh
of bash fib.sh

If I compile the above source code, will a new file be created? If so, which file?
Nee want het is een script die alleen execute permissies nodig heeft om uit te voeren.

Take relevant screenshots of the following commands:

- Compile the source files where necessary
- Make them executable
- Run them



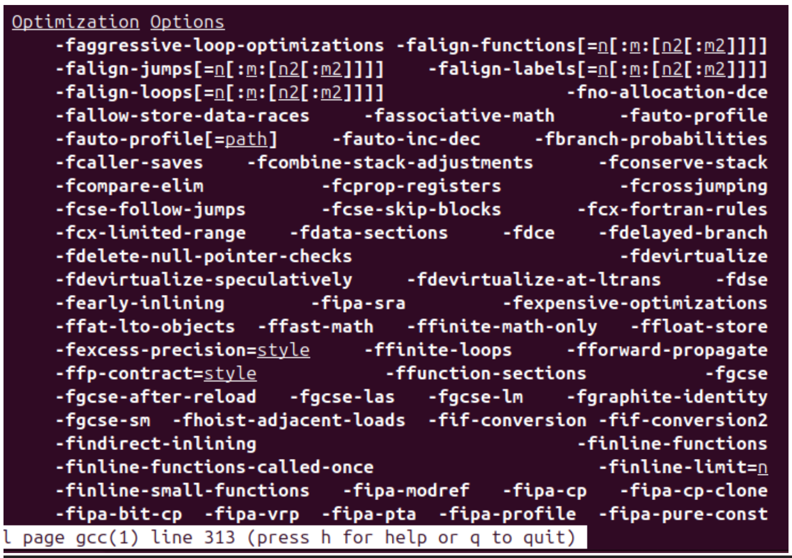
```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.39 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.71 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 6666 milliseconds


root@helpdesk:/mnt/hgfs/Ubuntu Share/code#
```

- Which (compiled) source code file performs the calculation the fastest?
  C is het snelst

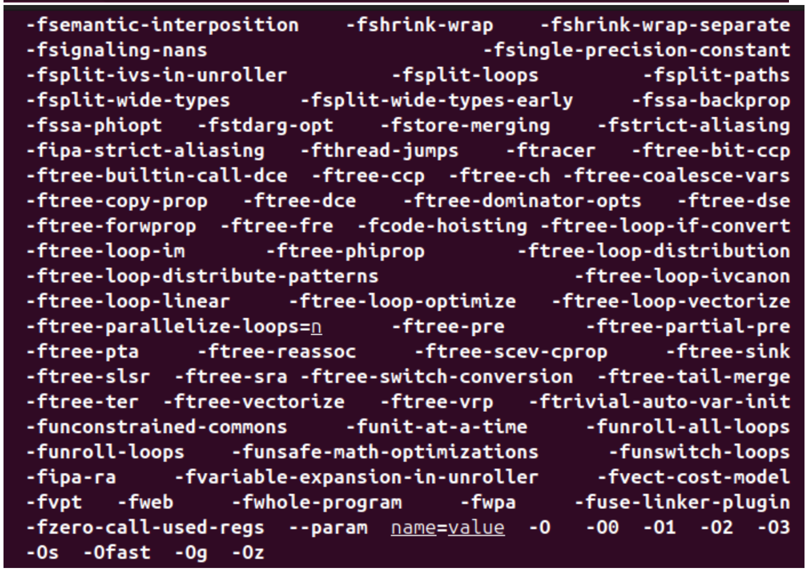
Assignment 4.4: Optimize

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.
In de man gcc staan een aantal optimalization opties zoals -O, -O0, -O1, -O2, -O3, -O4, Os -Ofast, -Og en -Oz

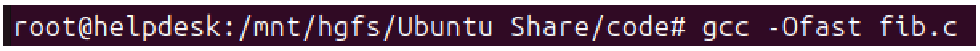


```
Optimization Options
    -faggressive-loop-optimizations -falign-functions[=n[:m:[n2[:m2]]]]
    -falign-jumps[=n[:m:[n2[:m2]]]]    -falign-labels[=n[:m:[n2[:m2]]]]
    -falign-loops[=n[:m:[n2[:m2]]]]                 -fno-allocation-dce
    -fallow-store-data-races      -fassociative-math      -fauto-profile
    -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
l page gcc(1) line 313 (press h for help or q to quit)
```



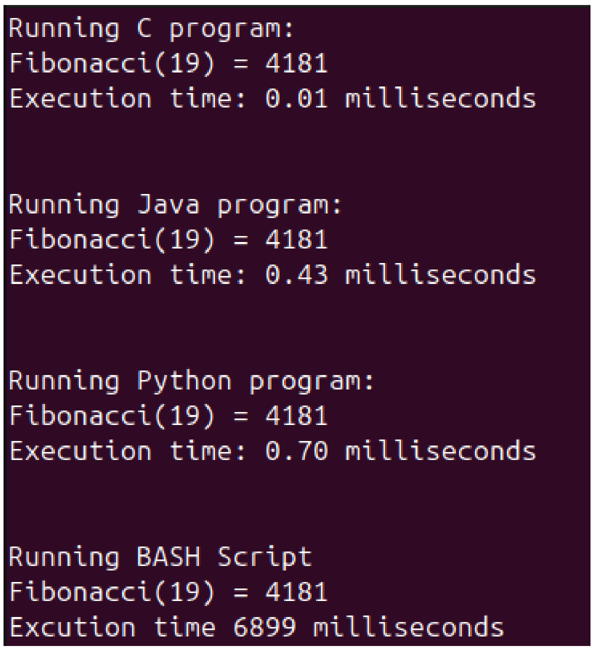
```
 -fsemantic-interposition    -fshrink-wrap    -fshrink-wrap-separate
 -fsignaling-nans                          -fsingle-precision-constant
 -fsplit-ivs-in-unroller          -fsplit-loops            -fsplit-paths
 -fsplit-wide-types       -fsplit-wide-types-early      -fssa-backprop
 -fssa-phiopt    -fstdarg-opt    -fstore-merging    -fstrict-aliasing
 -fipa-strict-aliasing    -fthread-jumps     -ftracer    -ftree-bit-ccp
 -ftree-builtin-call-dce   -ftree-ccp   -ftree-ch -ftree-coalesce-vars
 -ftree-copy-prop    -ftree-dce     -ftree-dominator-opts     -ftree-dse
 -ftree-forwprop   -ftree-fre   -fcode-hoisting -ftree-loop-if-convert
 -ftree-loop-im         -ftree-phiprop           -ftree-loop-distribution
 -ftree-loop-distribute-patterns                      -ftree-loop-ivcanon
 -ftree-loop-linear     -ftree-loop-optimize    -ftree-loop-vectorize
 -ftree-parallelize-loops=n        -ftree-pre           -ftree-partial-pre
 -ftree-pta       -ftree-reassoc      -ftree-scev-cprop       -ftree-sink
 -ftree-slsr   -ftree-sra -ftree-switch-conversion   -ftree-tail-merge
 -ftree-ter   -ftree-vectorize    -ftree-vrp    -ftrivial-auto-var-init
 -funconstrained-commons       -funit-at-a-time        -funroll-all-loops
 -funroll-loops      -funsafe-math-optimizations        -funswitch-loops
 -fipa-ra         -fvariable-expansion-in-unroller        -fvect-cost-model
 -fvpt    -fweb       -fwhole-program       -fwpa       -fuse-linker-plugin
 -fzero-call-used-regs   --param  name=value   -O    -O0   -O1   -O2   -O3
 -Os   -Ofast   -Og   -Oz
```

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



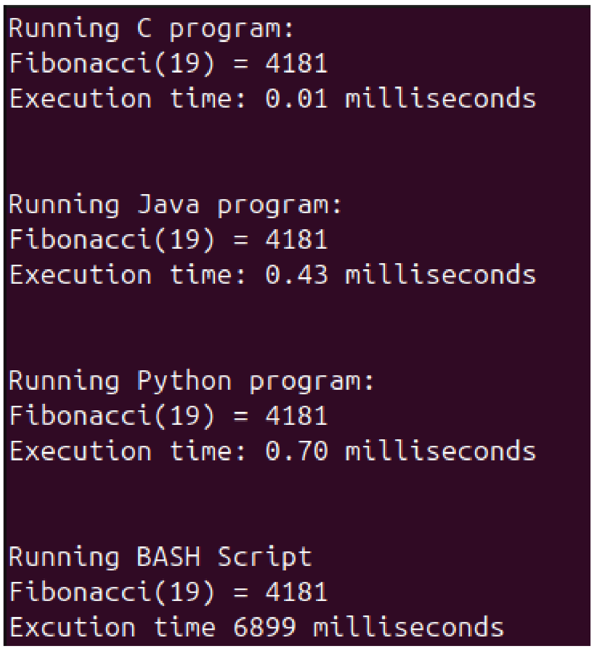
```
root@helpdesk:/mnt/hgfs/Ubuntu Share/code# gcc -Ofast fib.c
```

c) Run the newly compiled program. Is it true that it now performs the calculation faster?
Hij voert hem nogsteeds uit in 0.01 milliseconde dus ik weet niet of het sneller is



```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.43 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.70 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 6899 milliseconds
```

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.
Dit heb ik in de les gedaan dus ik weet niet meer hoe ik het heb gedaan



```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.43 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.70 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 6899 milliseconds
```

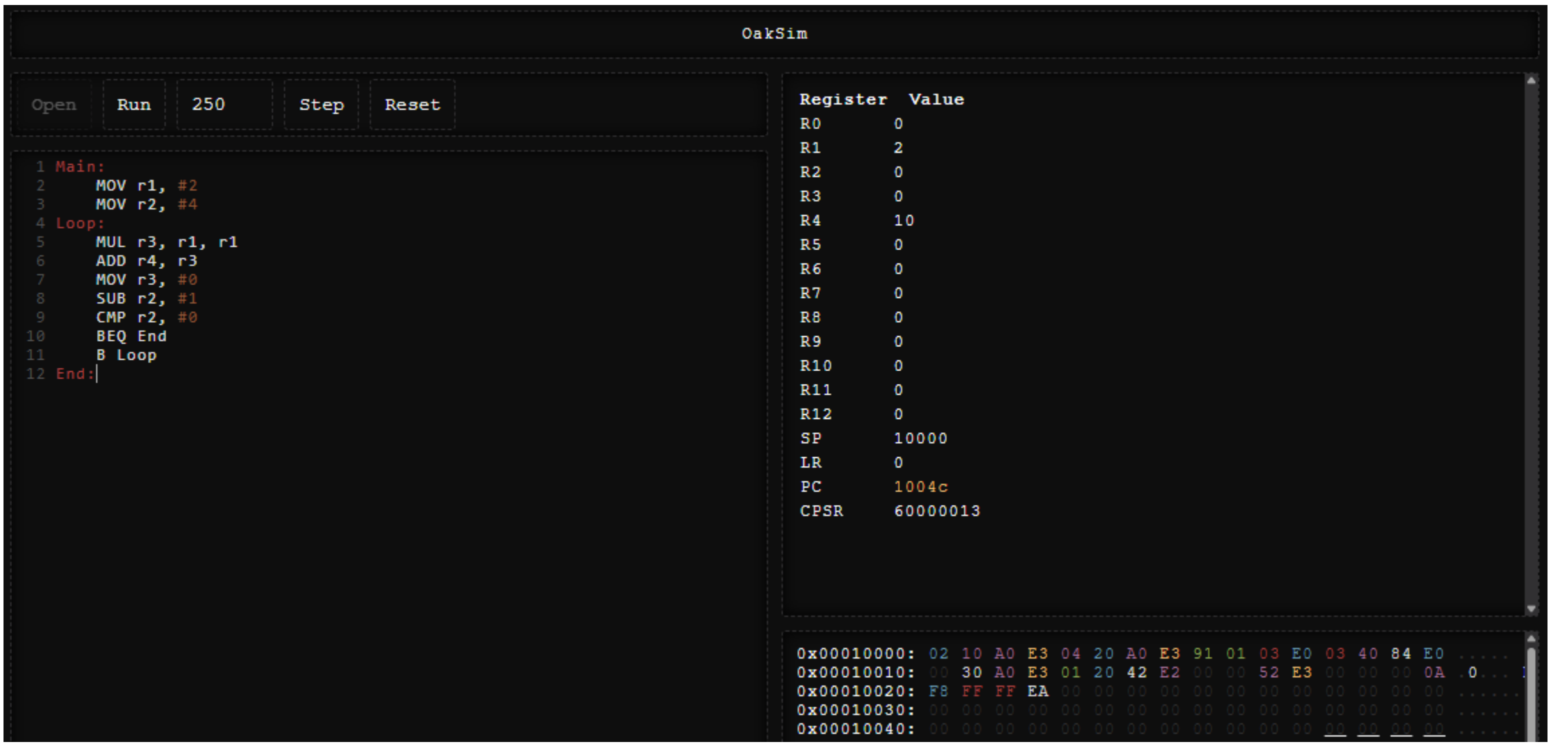
Assignment 4.5: More ARM Assembly

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.


OakSim

Open Run 250 Step Reset

```
Main:
    MOV r1, #2
    MOV r2, #4
Loop:
    MUL r3, r1, r1
    ADD r4, r3
    MOV r3, #0
    SUB r2, #1
    CMP r2, #0
    BEQ End
    B Loop
End:
```

```
Register  Value
R0        0
R1        2
R2        0
R3        0
R4        10
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
R11       0
R12       0
SP        10000
LR        0
PC        1004c
CPSR      60000013
```

```
0x00010000: 02 10 A0 E3 04 20 A0 E3 91 01 03 E0 03 40 84 E0 .....
0x00010010: 00 30 A0 E3 01 20 42 E2 00 00 52 E3 00 00 00 0A .0...
0x00010020: F8 FF FF EA 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 ......
```

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