RUHR
UNIVERSITÄT
BOCHUM

RUB

Name: **Tim Barsch**
Student ID: **108019210718**
Assignment Number: **3**
Date: November 10, 2021
Course: **ARM Processors for Embedded Cryptography**

# Text Assignments

## 1.1 A program on a P must always end in an infinite loop. Describe what would happen if the infinite loop was forgotten.

The processor didn't know when the program is ended and the processor will run into undefined instruction. That will properly cause an error and the processor will panic.

## 1.2 Explain in your own words the concepts of little-endian and big-endian memory.

The concept of endianess describes the memory layout of saved values. So like the name suggest, in little-endian is the least significant bit saved at the end and in big-endian the most significant bit.

## 1.3 Which role does the stack play when writing a function in ARM Assembly? What does the programmer have to ensure when using the stack in a function?

The stack is used to store values of registers without using the registers itself. So if a programmer call a function and overwrite some or all preserved registers in this function than the programmer has to restore this values at the end of the function. Otherwise the processor state of the caller function is now working on corrupt values. That mean the programmer has to ensure that the stack is leaved as it was found. A stack is also necessary to store the values of the link register outside of the function to ensure that a jump to the caller function is possible.

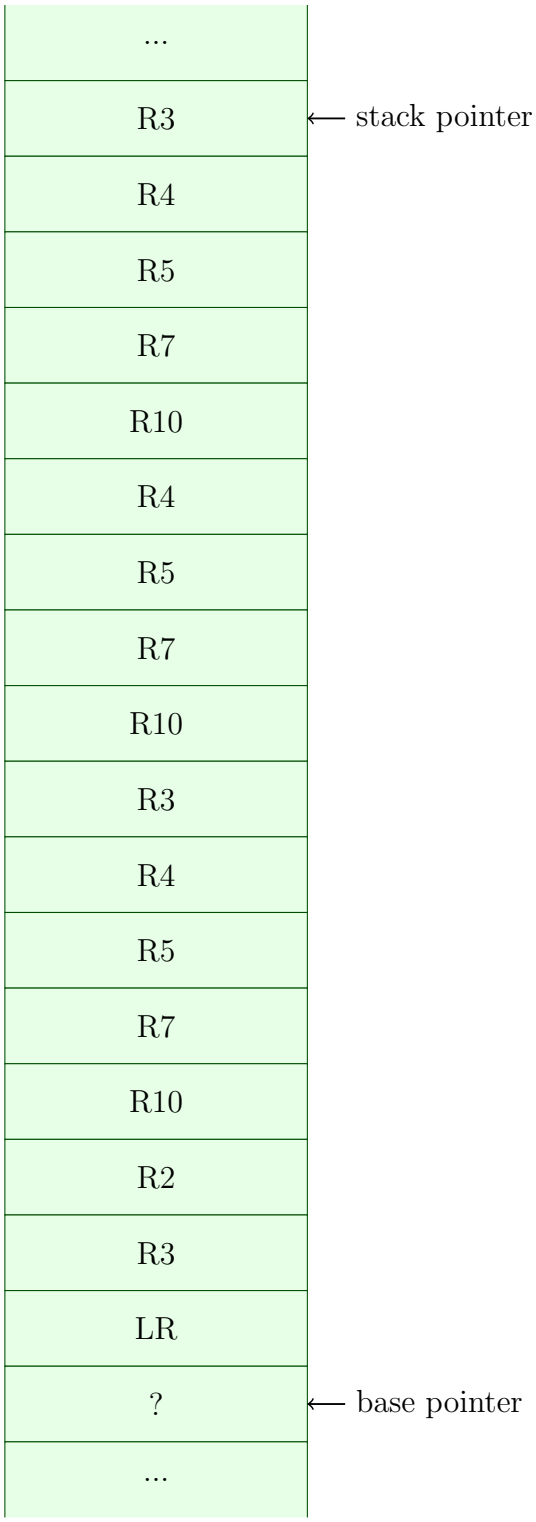
## 1.4 Why is PUSH R0, R1 faster than PUSH R0; PUSH R1?

The `PUSH` instruction of Cortex M4 needs $1 + n$ cycles to perform, where $n$ is the amount of registers to push on the stack, because the state registers is updated. So the first instruction just need 3 clock cycles, but the second one need 4 clock cycles.

## 1.5 With your knowledge about calling conventions, explain why exactly recursion is so expensive.

Recursion is so expensive because all used registers are pushed on the stack at each level of the algorithm. So if an instance of the algorithm has depth of ten and use just one preserved register, ten times there is 32-bit value stored and 2 clock cycles used. Additional to the pushes, this ten values have to popped again. A `POP` instruction uses $1 + n + p$ clock cycle, where $n$ is the amount of registers to pop and $p$ the number of cycles required for a pipeline refill. So in the end we use at worst 50 (20 through `PUSH`, 30 through `POP`) clock cycles just to push and pop.

## 1.6 Stack understanding



# Reversing Assignments

## 2.1

The return value for input $R0 = 11;\ R1 = 3$ is 2 and in general the function reduce the first input lower than the second input by subtracting the second from the first iterative. This operation is similar to modulus.

## 2.2

The return value for input $R0 = 42; R1 = 35$ is 7 and in general the function reduce both input values interactive from each other until they are equal. This operation is the euclidean algorithm.

## 2.3

This function call will return 11 and in general calculate this function the average value of the array elements rounded to zero.

## 2.4

The return value for input $R0 = 7; R1 = 4$ is 2401 and in general is this function calculating $R0^{R1}$.

## 2.5

This function return 1 if the array is in descending order, otherwise 0. So the return value for this call is 0.