Lesson2 Lab1

This lab requires compiling and linking two C files and a header file without using an integrated development environment (IDE). Therefore, writing custom startup and linker scripts is necessary, as will be demonstrated in this report.

The objective of the program was to transmit a text using the UART communication protocol on the arm926ej-s board and display that text on the terminal.

File1 -> uart.c

```c
#include"uart.h"

//defining the pointer to the UART register in the MC
#define UART0DT *((volatile unsigned int*)((unsigned int*)0x101f1000))

void Uart_sendData(unsigned char* P_tx_string)
{
    while(*P_tx_string != '\0') //looping in the array
    {
        UART0DT = (unsigned int)(*P_tx_string);//sending the characters to the register
        P_tx_string++; //increment the pointer to get the next character
    }
}
```

File2 -> uart.h

```c
#ifndef UART_H_
#define UART_H_

void Uart_sendData(unsigned char* P_tx_string);

#endif
```

File3 -> app.c

```c
#include "uart.h"

unsigned char name[100] = "Mohanad";

int main(int argc, char const *argv[])
{
    Uart_sendData(name);
    return 0;
}
```

Once the C files are compiled into object files, an assembly startup code is written to initialize the stack and call the main function.

Startup.s

```
.global reset
reset:
    ldr sp, = stack_top
    bl main
stop: b stop
```

Once the assembler is called to convert the startup code into an object file, it is necessary to link all the object files together to get the final file.

To guide the linker and initialize the memory according to specific requirements, a linker script is needed.

```
ENTRY(reset)
MEMORY
{
    Mem (rwx) : ORIGIN = 0x0000000 , LENGTH = 64M
}

SECTIONS
{
    . =. +0x10000;
    startup . :
    {
        startup.o(.text)
    }> Mem
    .text :
    {
        *(.text)
    }> Mem
    .data :
    {
        *(.data)
    }> Mem
    .bss :
    {
        *(.bss)
    }> Mem
    . = . + 0x1000;
    stack_top = . ;
}
```

Output

uart.o symbols:

```
$ arm-none-eabi-nm.exe uart.o
00000000 T Uart_sendData
```

app.o symbols:

```
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D name
         U Uart_sendData
```

.elf file symbols:

```
$ arm-none-eabi-nm.exe learn_in_depth.elf
0001000c T main
0001008c D name
00010000 T reset
000110f0 D stack_top
00010008 t stop
0001003c T Uart_sendData
```

The elf file is then converted to binary file and simulated by the qemu tool:

```
compu magic@DESKTOP-6HHQ8JH MINGW32 /e/Courses/Learn_in_Depth_Embedded_system_diplome/labs/Unit_3/lab1
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn_in_depth.bin
Mohanad
```

Same output using "Hi my name is Mohanad" as another input text:

```
compu magic@DESKTOP-6HHQ8JH MINGW32 /e/Courses/Learn_in_Depth_Embedded_system_diplome/labs/Unit_3/lab1
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn_in_depth.bin
Hi my name is Mohanad
```