Lab 4 - Vector Table and Startup

Purpose
To complete the base configuration of the STM32G071RB microcontroller.

Background
Previously, we compiled our basic embedded application at the command prompt. Now we switch to configuring CLion for embedded development adding changes needed to compile an application for our Nucleo board.

Setup Tasks

1. Download and install 'STM32CubeProgrammer' and 'STM32CubeMX'
2. Open CLion and create a new 'C Executable' Project.

   a. close any prior project first, if necessary
   b. give the project an appropriate name, like 'lab4'
   c. select C23 as the version of C

3. Copy the **main.c** and the linker definition file (i.e. **stm32.ld**) you created in lab 3 to the new folder.
4. Create a new Git repository for the project.

   a. located under the Version Control dropdown.

5. Configure the ARM toolchain.

   a. located under File->Settings->Build, Execution, Deployment->Toolchains
   b. use the MinGW template as a starting point
   c. change the C Compiler to the 'arm-none-eabi-gcc.exe' compiler
   d. change the C++ Compiler to the 'arm-none-eabi-g++.exe' compiler
   e. change the Debugger to the 'arm-none-eabi-gdb.exe' debugger
   f. the above files are located in the folder where the GNU ARM Embedded Toolchain was installed (e.g. C:\Program Files (x86)\GNU Arm Embedded Toolchain\bin)

```
Name:               ARM
Toolset:            C:\msys64\mingw64
CMake:              Bundled
Build Tool:         Detected: ninja
C Compiler:         ...\arm-none-eabi-gcc.exe
C++ Compiler:       ...\arm-none-eabi-g++.exe
Debugger:           ...\arm-none-eabi-gdb.exe
```

6. Configure CMake.

   a. located under File->Settings->Build, Execution, Deployment->CMake
   b. select the existing 'Debug' profile
   c. select the ARM toolchain from the dropdown
   d. rename the profile to 'Debug-ARM' (if needed)
   e. you may see errors like '**not able to compile a simple test program**' that you can ignore for now

Coding Tasks

1. Create a new file, called '**startup.s**'.
2. Add the following header, vector table definition and the basic startup assembler to the top of the file.

```
      .cpu cortex-m0plus
      .syntax unified
      .thumb
      .fpu softvfp

      .global _vectors, _start

      .section .vectors
      .align 2

      .type _vectors, %object
_vectors:
      .long _stack_top
      .long _start

      .rept 46
      .long 0
      .endr

      .size _vectors, . - _vectors

      .text
      .align 1

      .type _start, %function
_start:
      ldr r0, =_stack_top
      mov sp, r0

_done:
      b main

      .size _start, . - _start
```

3. Modify the linker definition file to include the vector table, the stack pointer and the new starting point.

```
MEMORY
{
    rom (rx)  : ORIGIN = 0x08000000, LENGTH = 128K
    ram (!rx) : ORIGIN = 0x20000000, LENGTH = 32K
}

_stack_top = ORIGIN(ram) + LENGTH(ram);

ENTRY(_start)

SECTIONS
{
    .text :
    {
        *(.vectors)
        *(.text)
    } > rom

...
```

4. Modify CMakeLists.txt to include configuration for embedded development.

```
cmake_minimum_required(VERSION 3.30)

set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_SYSTEM_PROCESSOR ARM)
set(CMAKE_SYSTEM_VERSION 1)

set(CMAKE_CXX_COMPILER arm-none-eabi-g++)
set(CMAKE_C_COMPILER arm-none-eabi-gcc)
set(CMAKE_ASM_COMPILER arm-none-eabi-gcc)
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

project(lab4 C CXX ASM)

set(CMAKE_C_STANDARD 23)
set(CMAKE_CXX_STANDARD 23)

add_definitions(-DDEBUG)

add_compile_options(-mcpu=cortex-m0plus -mthumb)
add_compile_options(-Wall -fomit-frame-pointer)

add_link_options(-T${CMAKE_SOURCE_DIR}/stm32.ld)
add_link_options(--specs=nosys.specs -nostdlib)

add_executable(lab4.elf main.c startup.s stm32.ld)
```

5. Use CLion to compile 🔨 a binary for the Nucleo board.
6. **Screenshot the CLion window** showing the successful compile and upload the image (PNG), along with this completed lab, to the corresponding NSCC BrightSpace assignment dropbox.
7. Read the resulting compiled binary with the STM32CubeProgrammer application.

> a. press 'Connect' to connect to the Nucleo board
> b. open the lab4.elf file
> c. ensure that the entire file contents can be seen in the displayed window

8. **Screenshot the STM32CubeProgrammer** window showing the **file contents** and upload the image (PNG), along with this completed lab, to the corresponding NSCC BrightSpace assignment dropbox.
9. Press the 'Download' button to upload the compiled binary to the Nucleo board.
10. Use the STM32CubeProgrammer to read the memory of the Nucleo board.

> a. select the 'Device memory' tab
> b. press the 'Read' button
> c. ensure that memory from 0x0800000 to 0x080000D0 can be seen in the displayed window

11. **Screenshot the STM32CubeProgrammer** window showing the **memory contents** and upload the image (PNG), along with this completed lab, to the corresponding NSCC BrightSpace assignment dropbox.

Questions

1. What is the 'vector table'? Where must it be located in memory?

A 'vector table' is a table that contains memory addresses.

ARM Cortex-M Address: 0x0000_0000

2. Which vector table entry tells the microcontroller where to begin executing code?

Reset Handler: This entry is where the first piece of code is executed after a reset
Address: 0x0000_0004