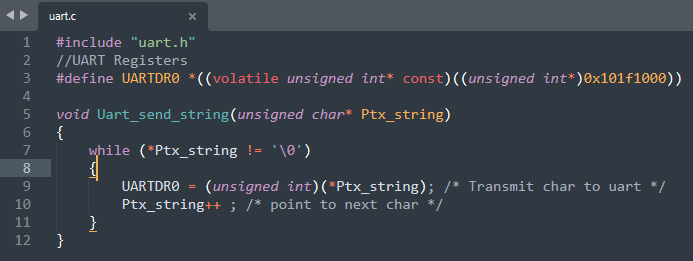
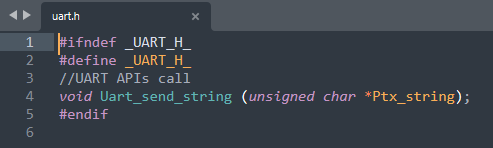
Lab1 : Build baremetal application on arm versatilepb

* C codes :-

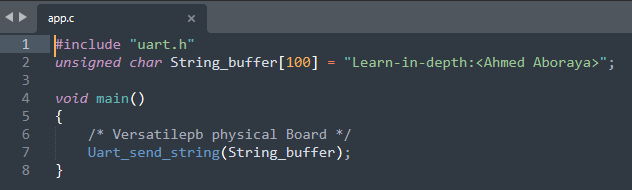
uart.c : definition of function



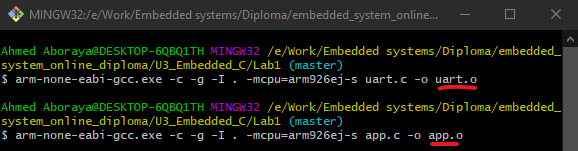
uart.h : declaration of the function

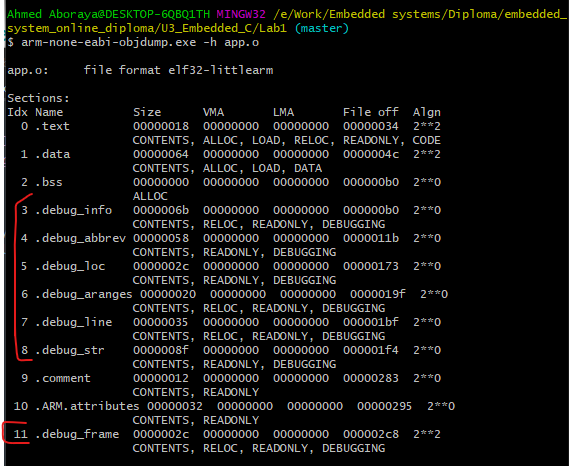


app.c : calling of the function



We gonna compile these c codes in terminal using ARM compiler …

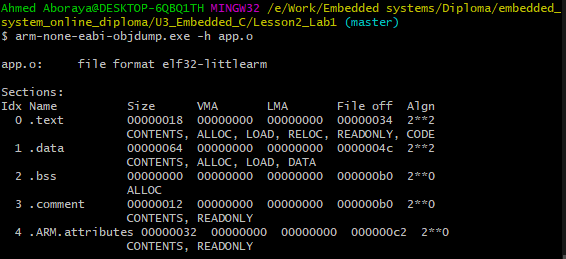
* generate object files: using ARM Toolchain
* “arm-none-eabi-gcc” GNU ARM-Cross-toolchain
* “-c” compile and assemble ,but don’t link
* “-g” include debug information in object file
* “-I .” include file in the current directory
* “-mpcu-arm926ej-s” get specific assembly code compatible with this processor
* “-o <file>” pace the output into <file>
* **Using ARM-Cross toolchain Bin Utilities (objdump)**



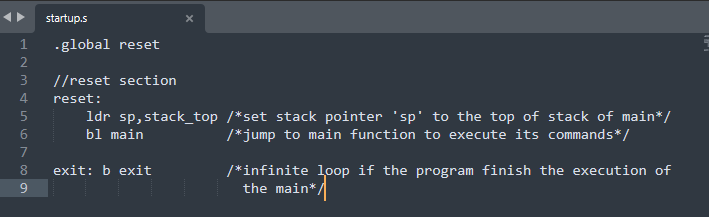
* **“arm-none-eabi-objdump”:** bin tool to display information fromobject <files>.
* **“-h”**: Display the contents of the section headers.
* **VMA:** Virtual Memory Address of the output section**.**
* **LMA:** Load Memory Address of the output section**.**
* Sections with red mark are **debug sections**.

After removing “-g” gcc parameter of debug info…

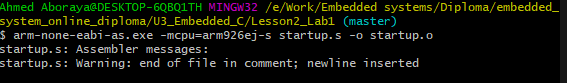
* **sections without debug :**



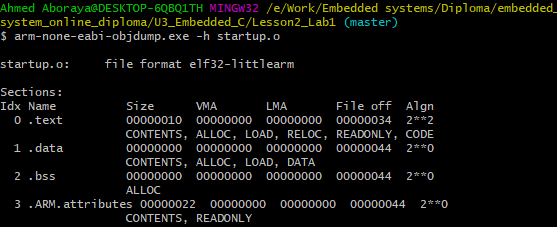
* **Writing a startup.s file:**

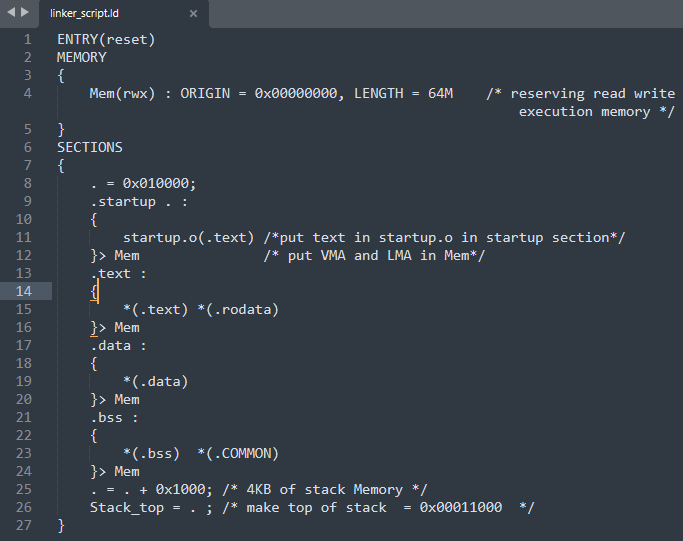


* generate object file: “compiling”



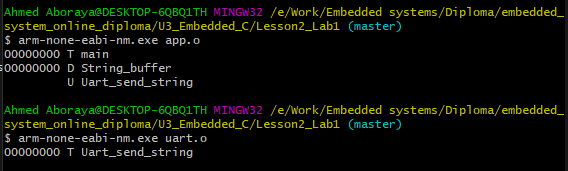
* **“arm-none-eabi-as”:** Arm tool to deal with and convert assembly files.
* **Analyzing startup.o :**



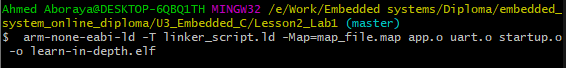
* Make linker script file:
* **“ENTRY”** Define the entry point of an application, this information will be in the header of final elf file.
* **“SECTIONS”** create different output sections in the final executable file.
* **“Attributes”** : “-r” read only section, “w” read/write section, **“x”** section contain

executable code .

* **read the symbols: (from .o files)**

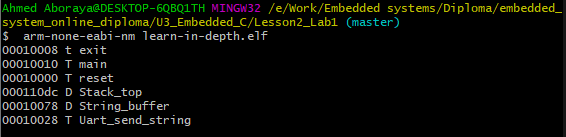


* **“arm-none-eabi-nm”:** nm is cross tool chain bin utility to read symbols.
* **Linking all the objects:**

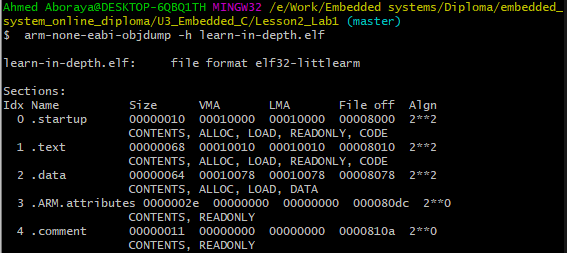


* **“arm-none-eabi-ld”:** ld is utility to linking all object files with linker script
* **“-T”(trace):** read linker script
* **“.map”:** file gives a complete listing of all code and data addresses for the final software image.
* **Analyze the executable file:**

**(reading symbols):**

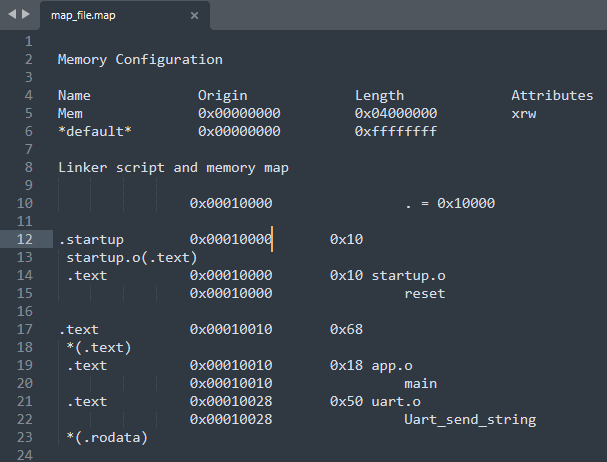


**(reading sections):**



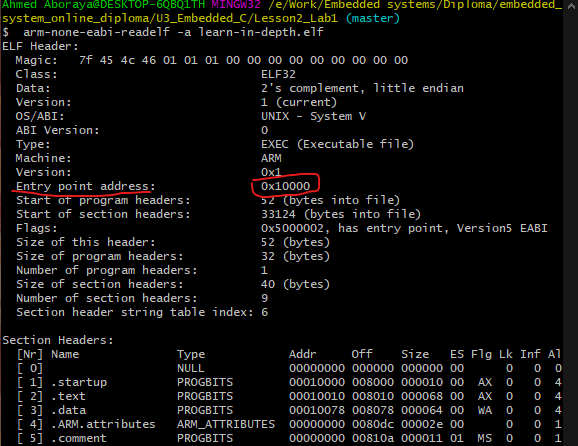
**(reading sections):**

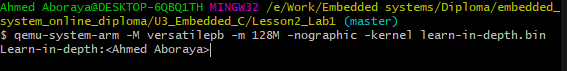
**(Map file):**



**Generate binary file:**



* read headers of elf file:
* Make sure of entry point address
* **run the program in the QEMU Simulator**
* **(“VersatilePB physical Board”)**



* **“-M (-machine) [name]”:** selects emulated machine.
* **”-m [size]”:** configure guest RAM, size: initial amount of guest memory.
* **“-nographic”:**disable graphical output and redirect serial I/O to console.
* **“-kernel”:** mode to communicate with processor directly without operating system (Baremetal).

Burn the code on this board…

And finally it works ^\_^