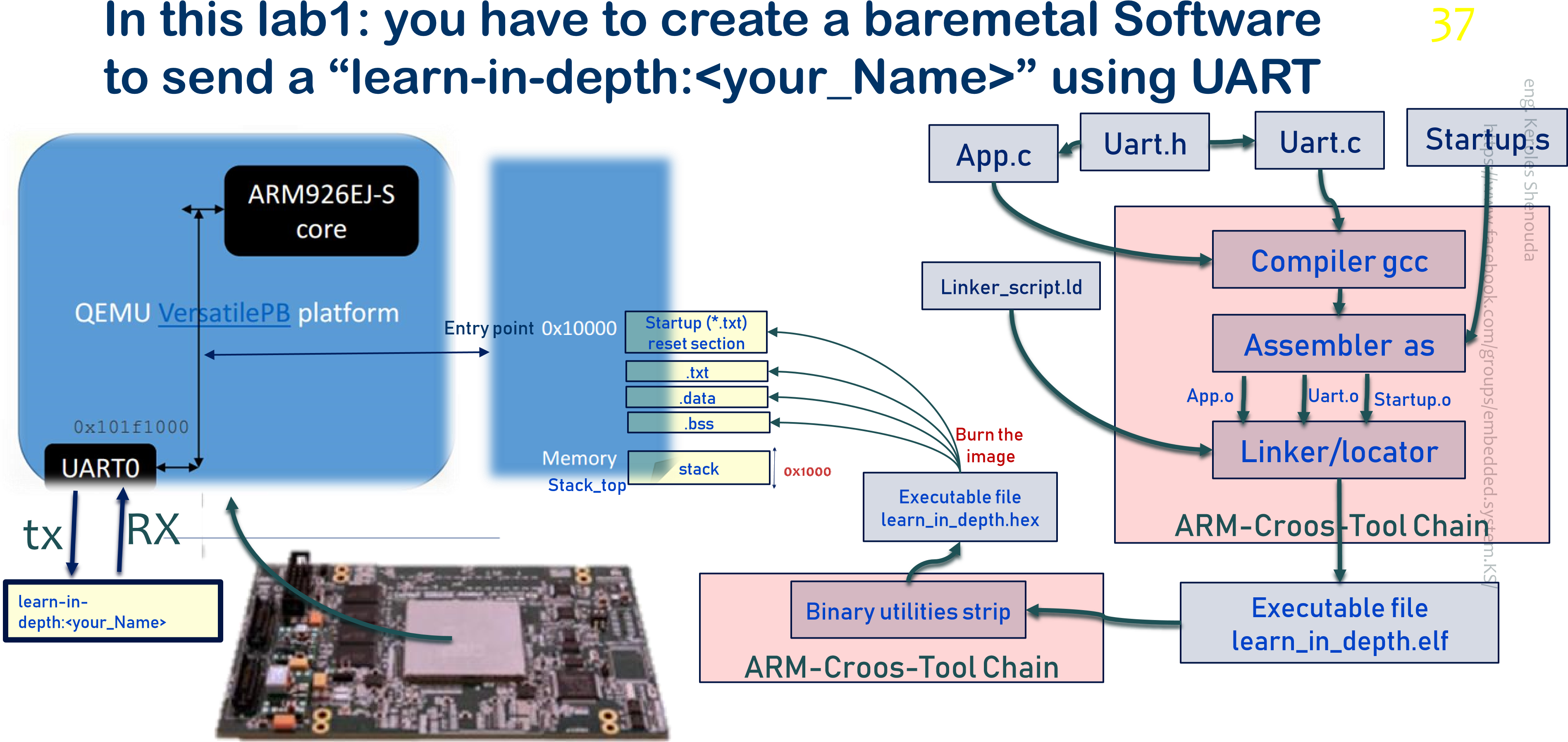
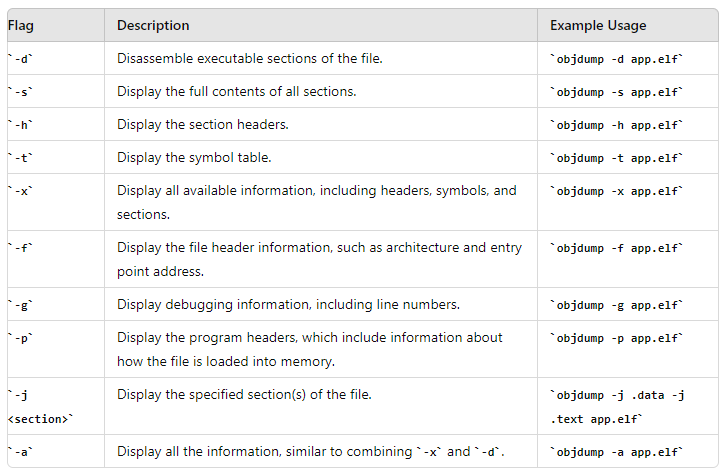
**In this lab1: you have to create a baremetal Software to send a “learn-in-depth:<your\_Name” using UART**

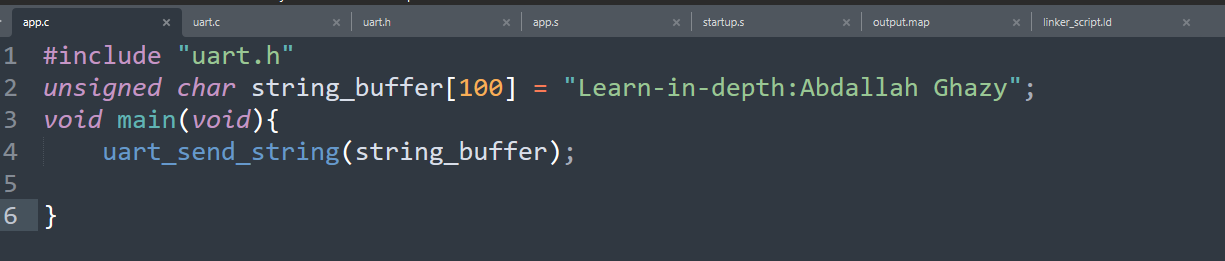


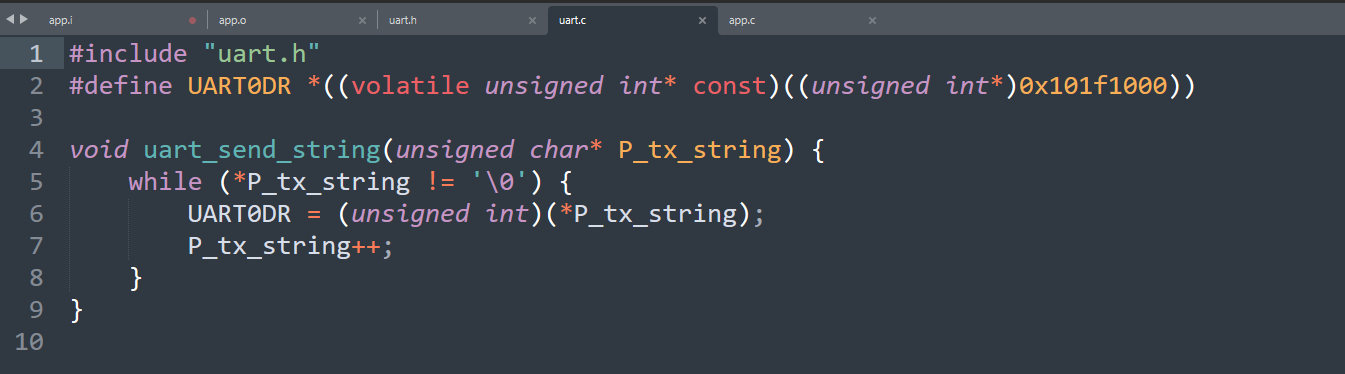
### The Most Important Flags Used with The Objdump

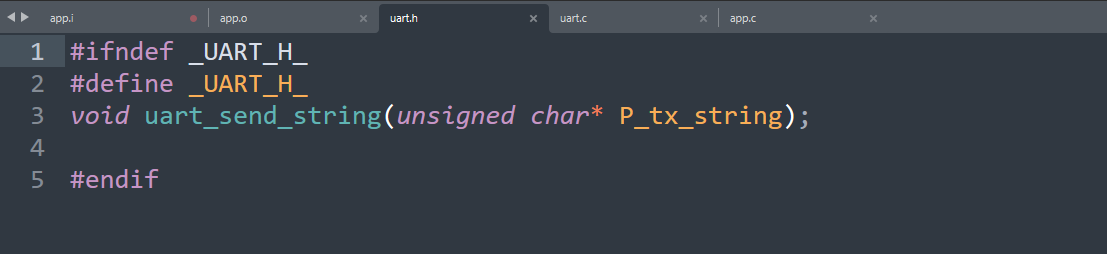


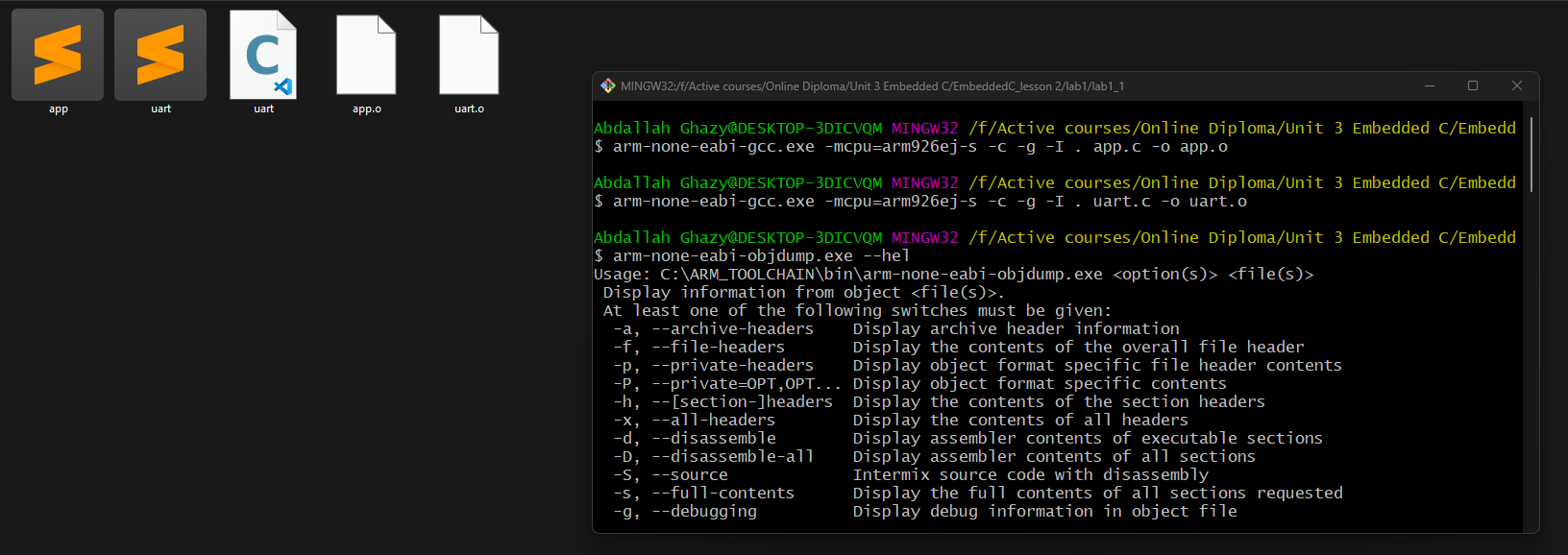
### c code files

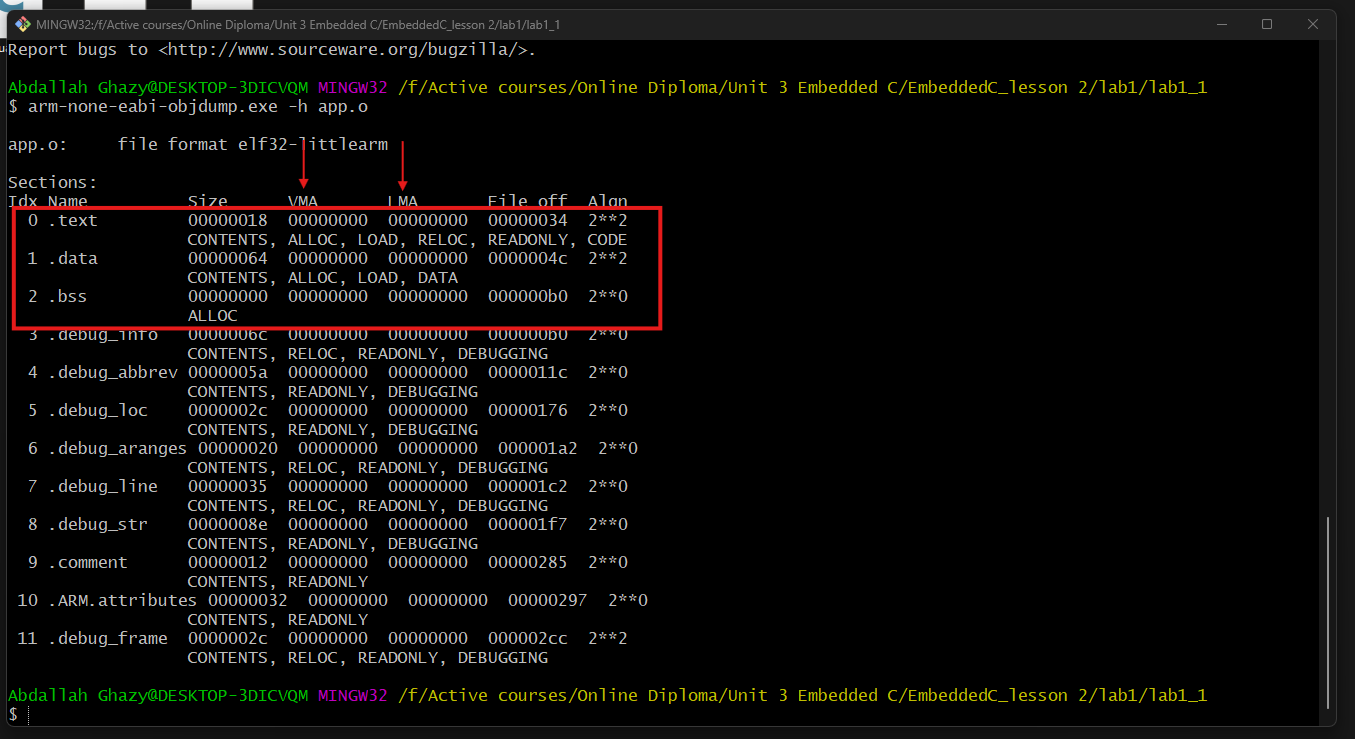
the address where the UART0 is mapped: 0x101f1000











Debug sections

**LMA (Load Memory Address)**

* **Definition**: The Load Memory Address is the address at which a section of code or data is loaded into memory during the linking or loading phase. It represents the address where the program is initially placed in memory before execution.

**Example**: For a section of code in a linker script, the LMA might specify that the code is to be loaded at address 0x08000000 in flash memory.

**VMA (Virtual Memory Address)**

* **Definition**: The Virtual Memory Address is the address used by a program or process within its virtual address space. It is the address that the program uses when it references memory. This address is abstracted from the physical memory addresses by the operating system’s memory management unit (MMU).

**Example**: In a memory-mapped file or a dynamically allocated memory region, the VMA is the address that the application code uses to access the memory.

### Executable file sections (.data, .bss and rodata)

When C code is compiled, the compiler places initialized global variables in the .data section. So just as with the assembly, the .data has to be copied from Flash to RAM

The C language guarantees that all uninitialized global variables will be initialized to zero.

When C programs are compiled, a separate section called .bss is used for uninitialized variables. Since the value of these variables are all zeroes to start with, they do not have to be stored in Flash. Before transferring control to C code, the memory locations corresponding to these variables have to be initialized to zero.

# .bss (block started by symbol)

# Interview trick

bss is not in flash as it is not have a value, we just reserve a section for it in ram by knowing its size and initialize this section by zero.

Since those variables do not have any intial values, they are not required to be stored in .data section (.data section is stored in Flash)

All uninitialized (global/static)variables are stored in .bss

# Read-only Data

GCC places global variables marked as const in a separate section, called .rodata. The .rodata is also used for storing string constants.

Since contents of .rodata section will not be modified, they can be placed in Flash. The linker script has to modified to accomodate this

### Load Location (LMA - Load Memory Address)

**Definition**:

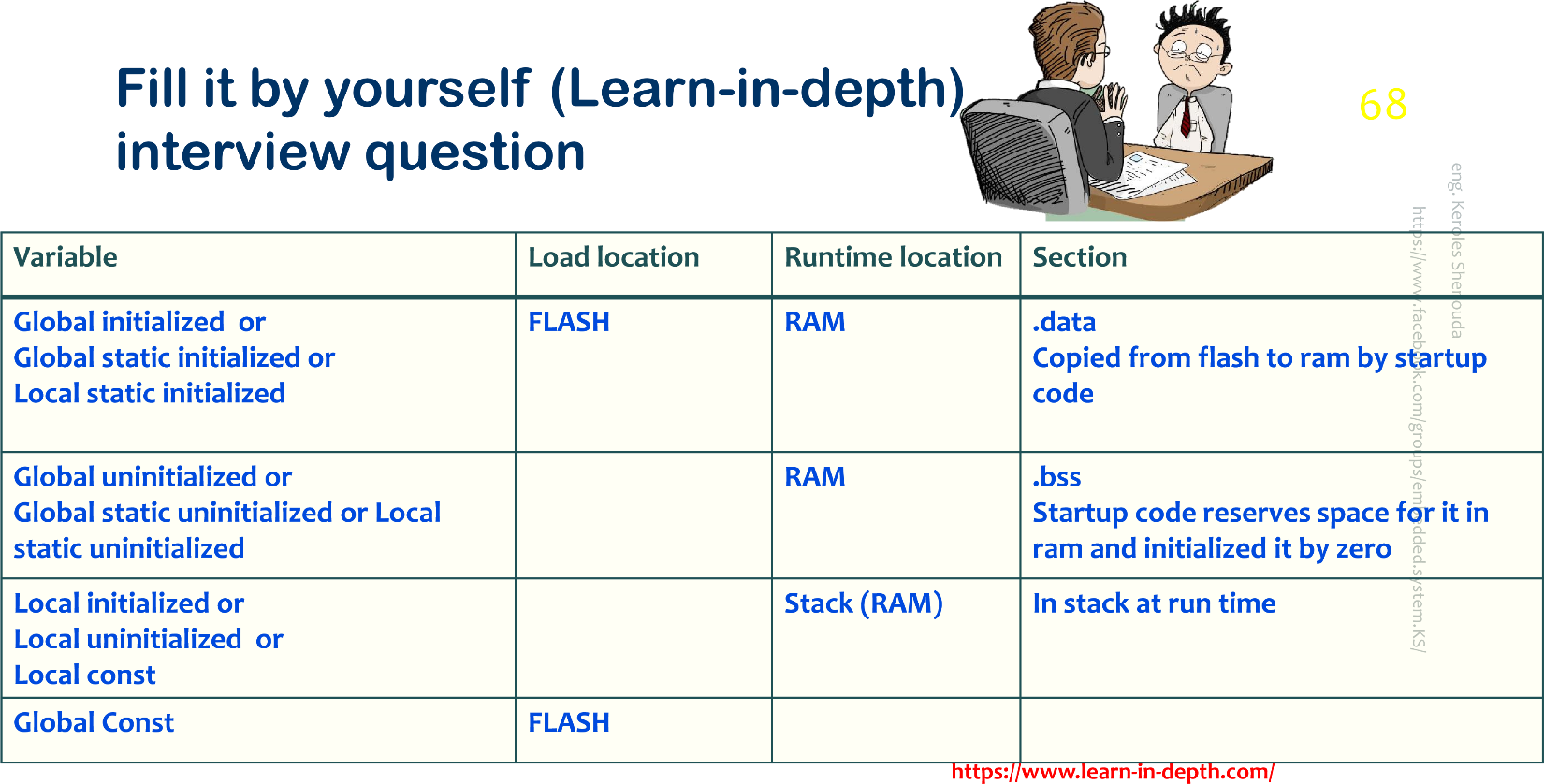
* The Load Location is the address where the program or data is loaded into memory during the initialization or loading phase. This is the physical memory address where the code or data is placed before the program starts executing.

**Usage**:

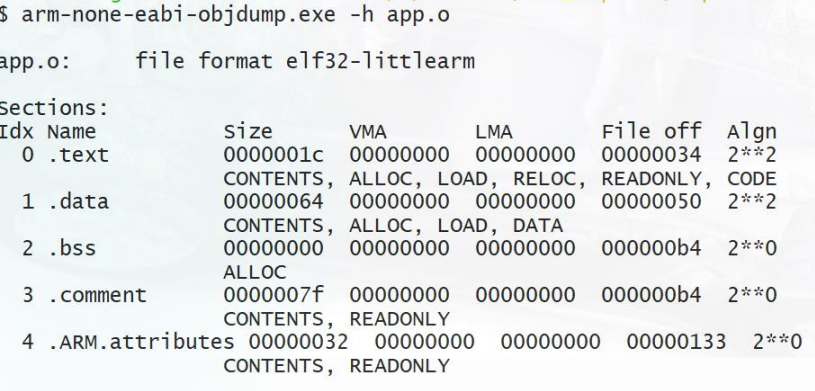
* **Embedded Systems**: In embedded systems, the Load Location is where the firmware is loaded into memory from non-volatile storage (e.g., flash memory) into RAM or another execution space.
* **Linker Scripts**: The Load Location is often specified in linker scripts to define where various sections of the code or data should be placed in memory.

### Runtime Location (VMA - Virtual Memory Address)

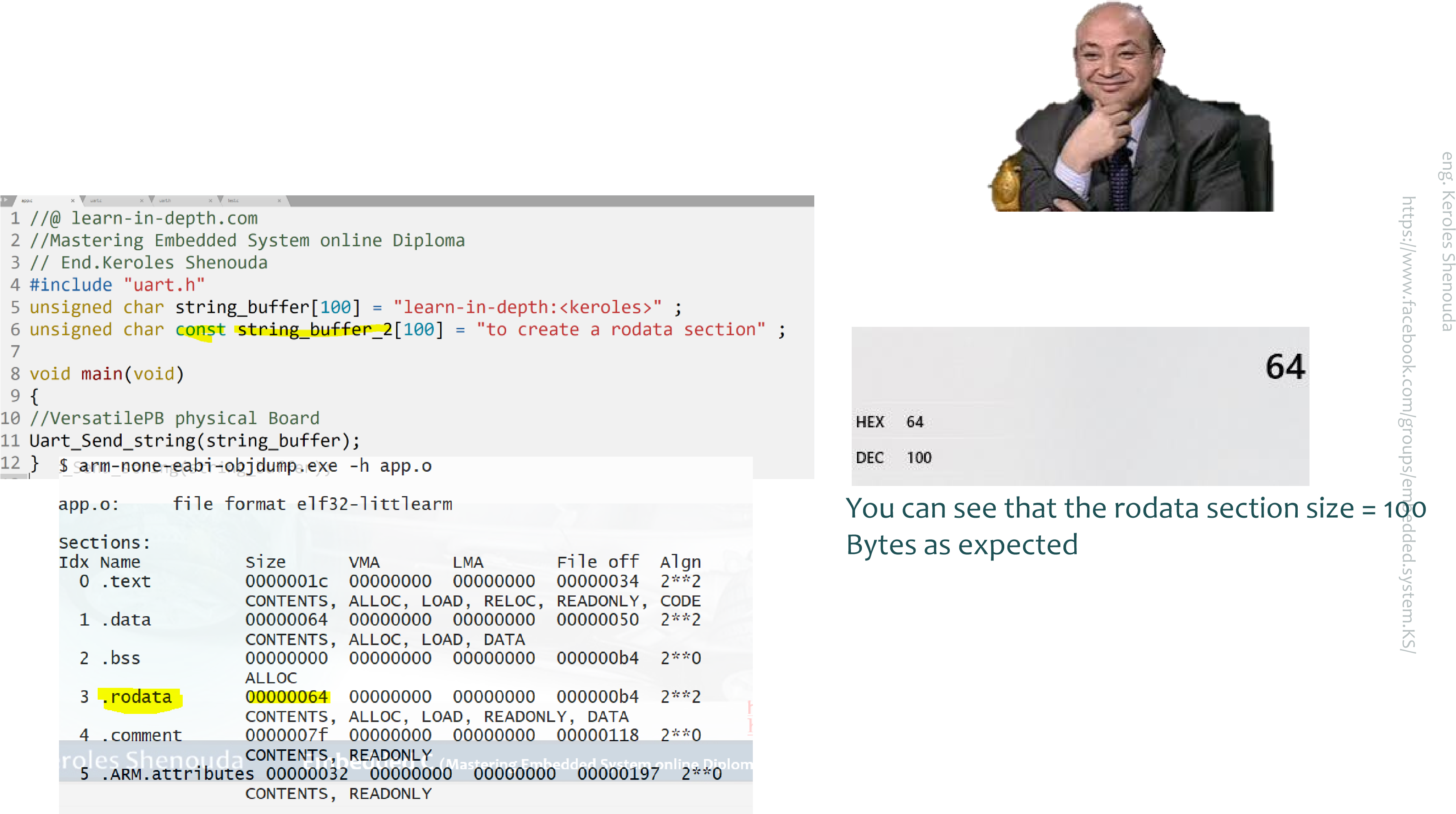
* **Definition**:
  + The Runtime Location refers to the address that the program uses during execution to access its code or data. This is often an address within the program's virtual memory space.
* **Usage**:
  + **Virtual Memory**: In systems with virtual memory, the Runtime Location is the address as seen by the program. The operating system's memory management unit (MMU) maps this virtual address to a physical memory address.
  + **Debugging**: During debugging, you might refer to the Runtime Location to understand where variables and functions are accessed during program execution.
* **Example**:
  + When a program is running, it may access a variable at a virtual address 0x20001000. This is its Runtime Location, which is mapped to a physical address by the operating system.



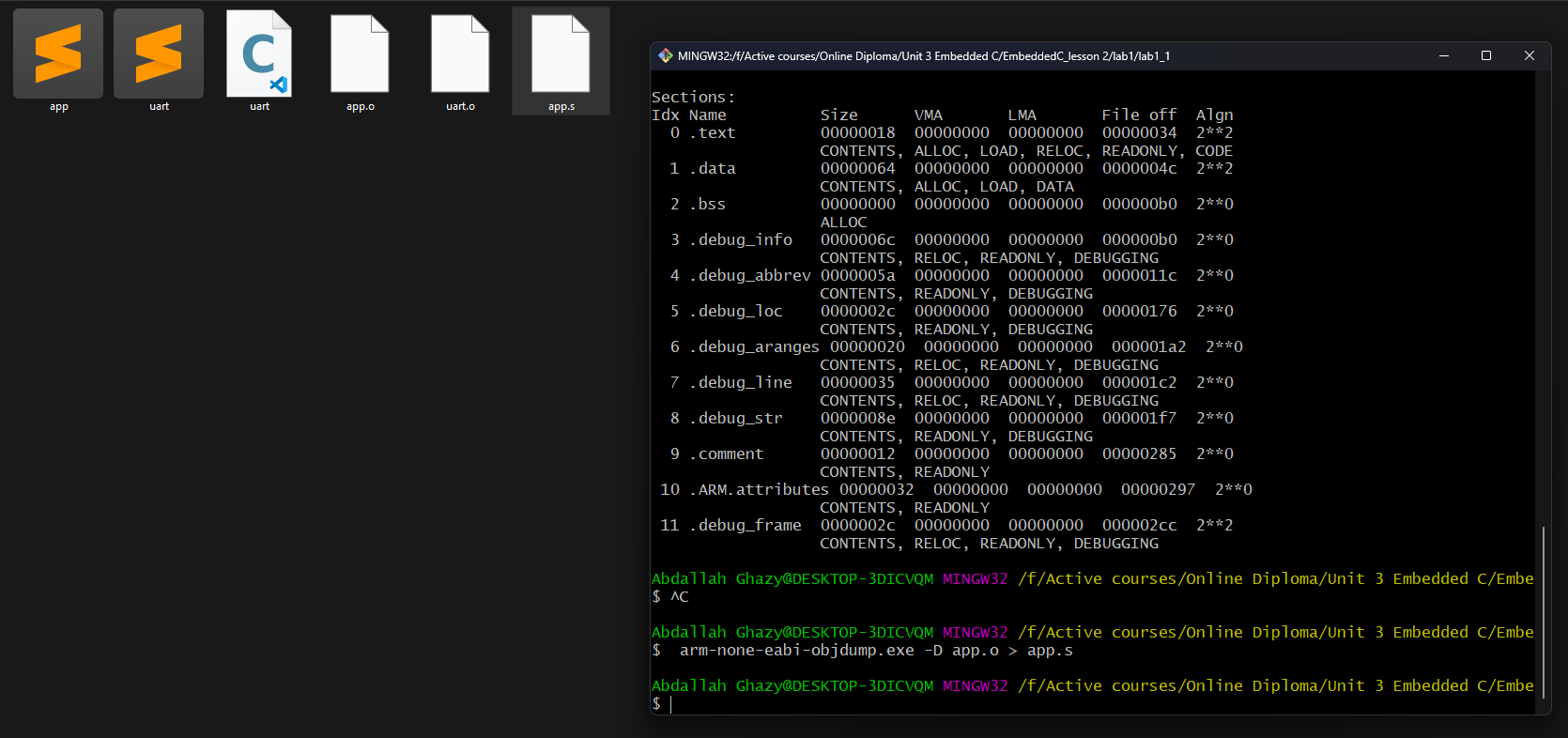
# Where is a .rodata section we didn’t find it, why and could you generate it now ?

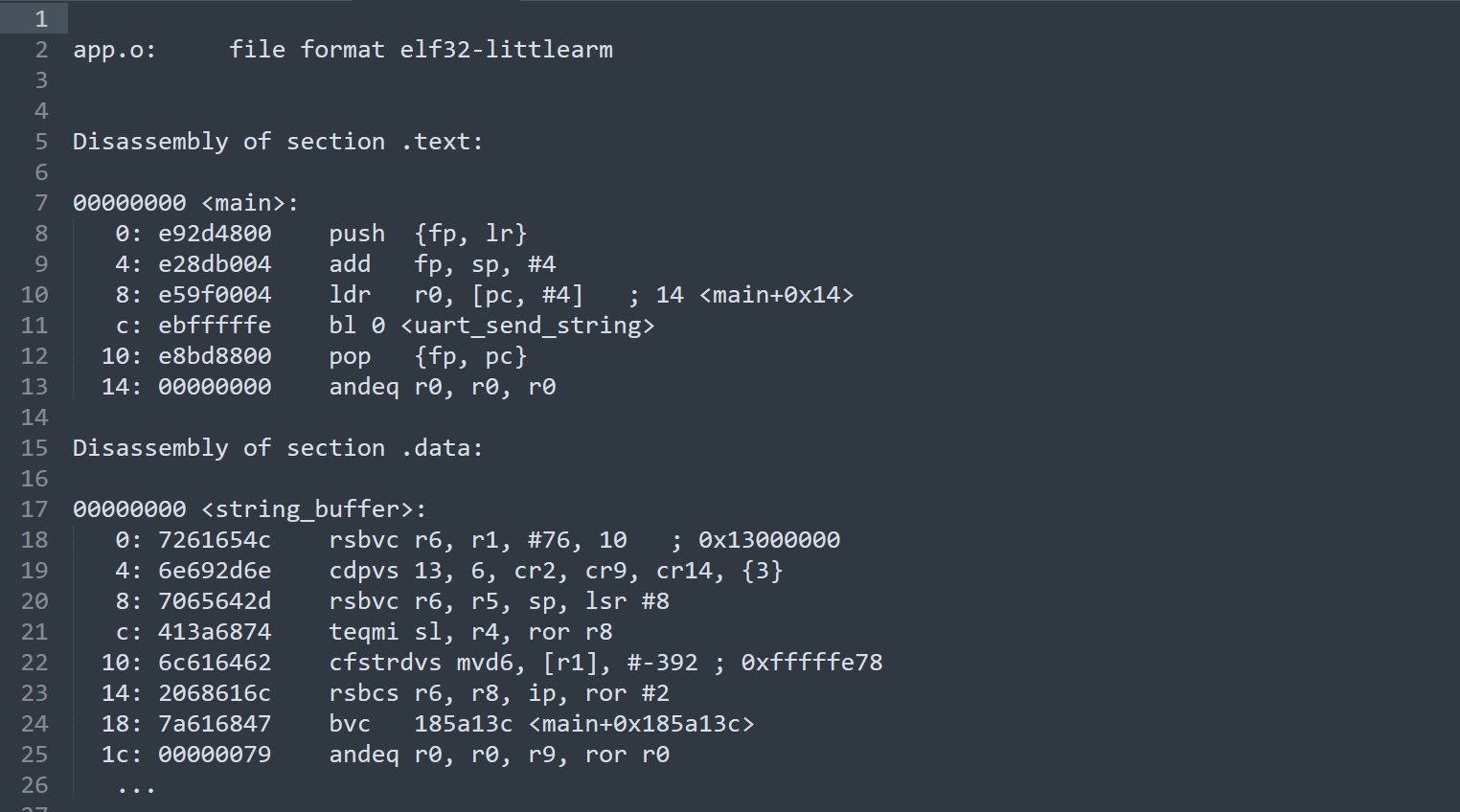


* GCC places global variables marked as const in a separate section, called .rodata. The .rodata is also used for storing string constants.



# Let us generate the disassembly file from the bin

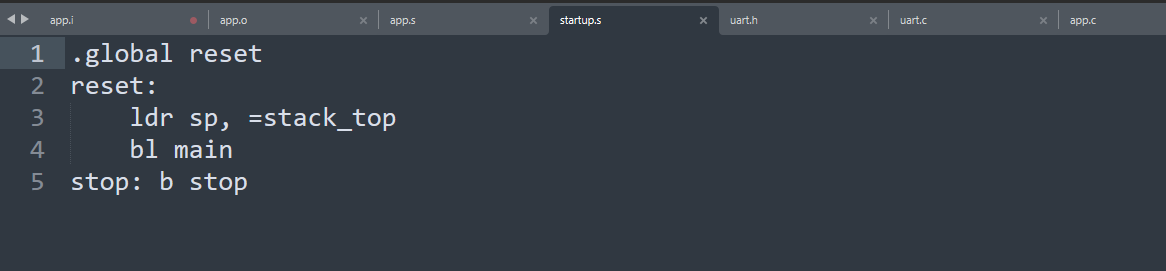


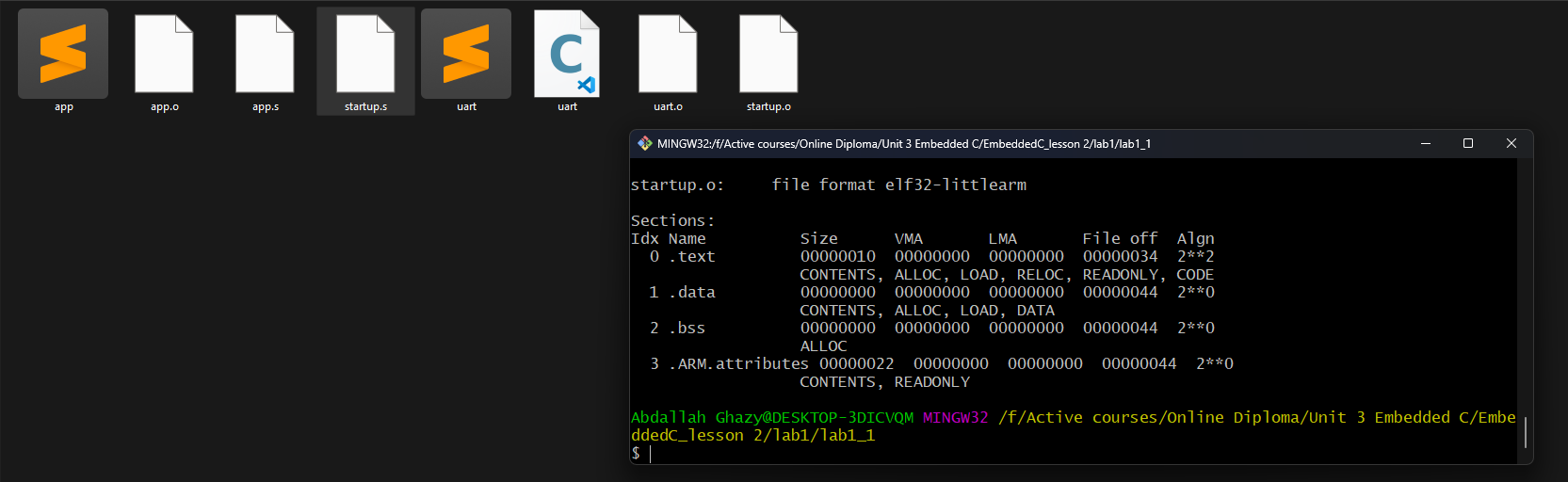


### In Lab1: We will write a simple startup:

1. Create a reset section and Call main( ).

2. Initialize Stack





### ****The Linker Script**** and ****Locator (Linker Locator)****

Locator (Linker Locator)

#### Definition

A **locator** (often referred to as a linker locator or memory locator) is not a separate tool but rather refers to the functionality or components within the linker that handle the placement and address calculation of code and data sections in memory.

#### Functions

1. **Address Calculation**:
   * Calculates and assigns addresses to code and data sections based on the linker script.
2. **Memory Management**:
   * Manages how sections are placed in specific memory regions and ensures they meet alignment requirements.
3. **Section Handling**:
   * Ensures that sections are located according to the directives in the linker script.

### Linker Script

#### Definition

A **linker script** is a text file used to control the behavior of the linker during the linking process. It specifies how different sections of the code and data should be arranged in memory and how they should be combined.

#### Functions

1. **Memory Layout**:
   * Defines memory regions and their properties (e.g., FLASH, RAM).
   * Specifies the starting addresses and sizes of different memory regions.
2. **Section Placement**:
   * Controls where different sections of the program (e.g., .text, .data, .bss) should be placed in memory.
   * Determines how sections from various object files are combined.
3. **Symbol Definitions**:
   * Can define symbols or variables that the linker will use.
4. **Alignment**:
   * Specifies alignment requirements for sections.

reflect exactly the memory resources and memory map of the target microcontroller

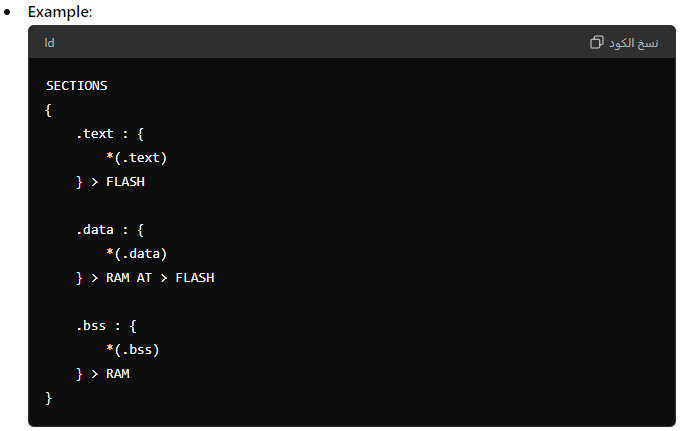
GNU linker script has the file extension \*.ld

You have to use the linker script at the linking phase by pass to the liker option –T

# Common Linker Script Commands

**SECTIONS**

* **Purpose**: Defines the layout of the sections in memory.
* **Usage**: Specifies how different sections (like .text, .data, .bss) are mapped to memory.



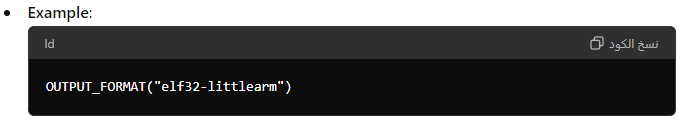
**ENTRY**

* **Purpose**: Specifies the entry point of the program.
* **Usage**: Defines where the execution starts.



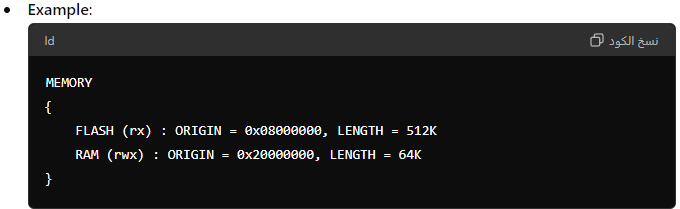
**OUTPUT\_FORMAT**

* **Purpose**: Sets the format of the output file.
* **Usage**: Specifies the format for the generated executable or object file.



**MEMORY**

* **Purpose**: Defines the memory regions available for use.
* **Usage**: Specifies the start, end, and size of different memory regions.



**PHDRS**

* **Purpose**: Defines program headers for executables.
* **Usage**: Specifies the type of segments for the executable file.



**AT**

* **Purpose**: Specifies the load address of a section.
* **Usage**: Indicates where a section should be loaded in memory.



**STARTUP**

* **Purpose**: Specifies the startup file.
* **Usage**: Defines the initialization routine or startup code to be linked.



**PROVIDE**

* **Purpose**: Defines a symbol with a default value.
* **Usage**: Useful for defining default values or placeholders.



**GROUP**

* **Purpose**: Groups multiple files or sections together.
* **Usage**: Ensures that all files or sections in the group are included in the output.



**INCLUDE**

* **Purpose**: Includes other linker script files.
* **Usage**: Modularizes linker scripts by including additional scripts.



**Summary**

* **SECTIONS**: Specifies the memory layout.
* **ENTRY**: Defines the entry point of the program.
* **OUTPUT\_FORMAT**: Sets the output file format.
* **MEMORY**: Defines memory regions.
* **PHDRS**: Specifies program headers.
* **AT**: Indicates the load address of a section.
* **STARTUP**: Specifies startup code.
* **PROVIDE**: Defines default values for symbols.
* **GROUP**: Groups files or sections together.
* **INCLUDE**: Includes additional linker scripts.

### Linker script command’s location counter

Linker symbol ‘.’ Dot

This dot is called “location counter” it is automatically address calculated by each section size

We can use it to track and define the memory layout boundaries υ

Also we can use it to specify specific address for specific section

Location counter should use only in sections command

# Linker script commands using >(vma) AT>(lma)

vma is specify relocatable section address in run-time located

Lma is specify relocatable section address in load-time located

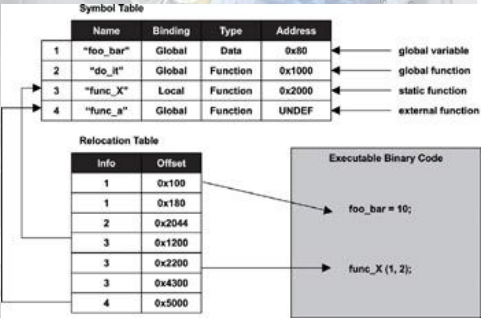
### Linker script (Symbols)

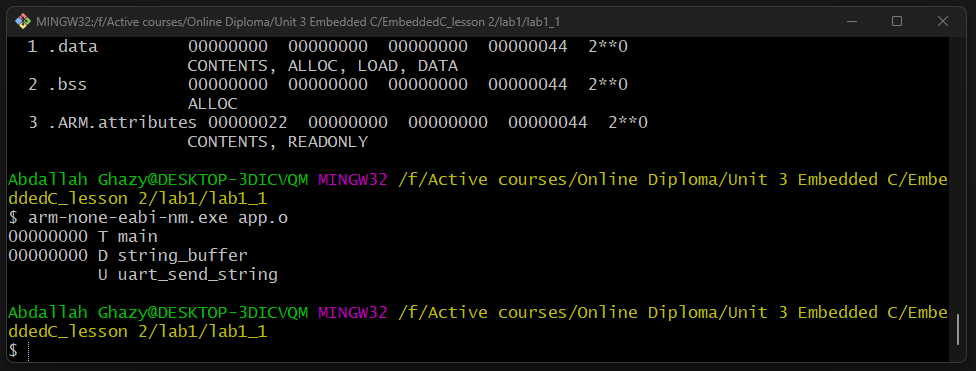
Symbol is the name of an address

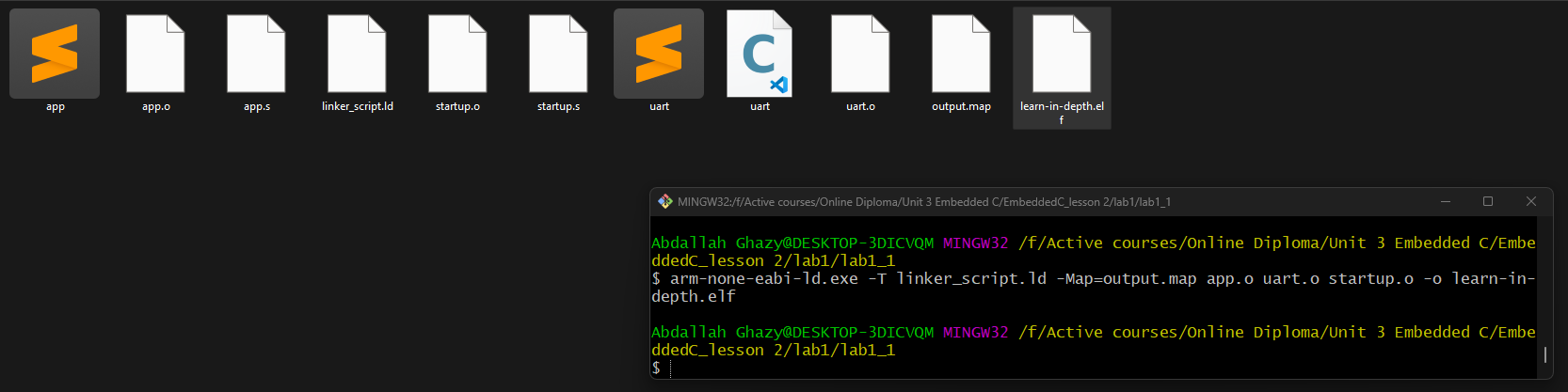
Symbol declaration is not equivalent to variable declaration.

Each object have its own symbol table, the linker is resolving the symbols between all obj files.

Symbol also is used to specify Memory layout boundaries

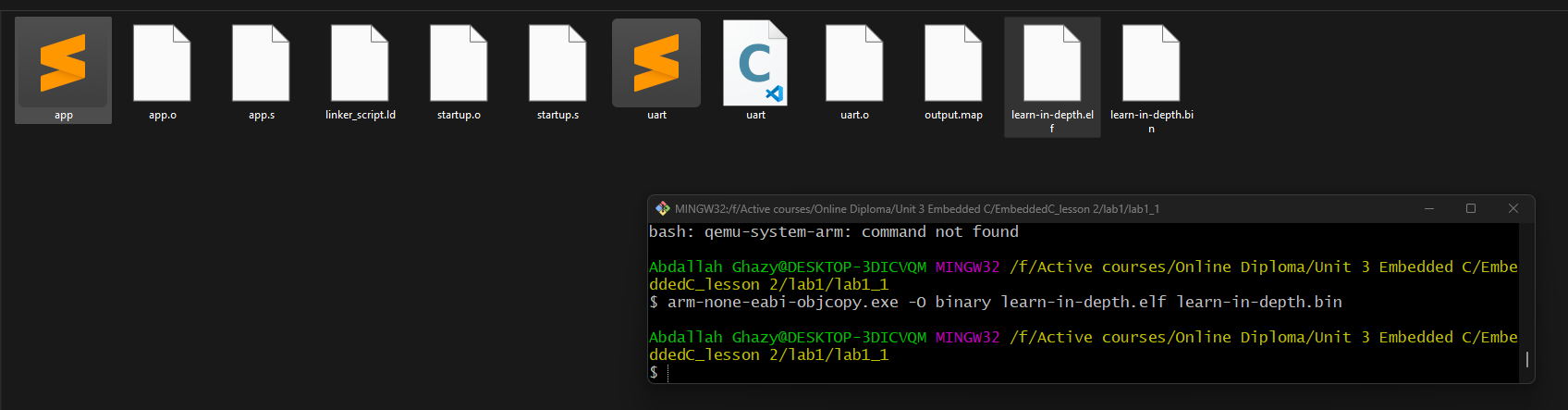






The .map file gives a complete listing of all code and data addresses for the final software image. It provides information similar to the contents of the linker script described earlier. However, these are results rather than instructions and therefore include the actual lengths of the sections and the names and locations of the public symbols found in the relocatable program

Generate binary file



# To run the program in the QEMU Simulator (“VersatilePB physical Board”)

