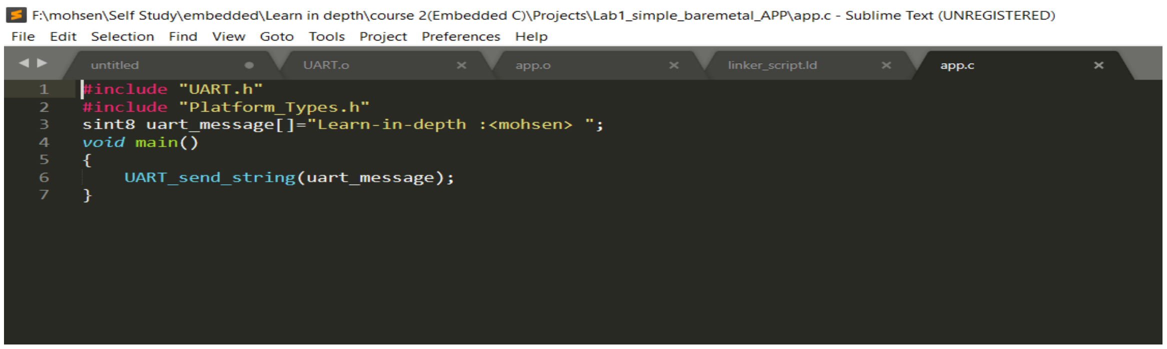
Lab1 Report

1 Introduction:

in this report I will build a simple bare-metal application that sends a string message using UART protocol in VersatilePB micro-controller chip which is based on arm926ej-s microprocessor, I will build everything from scratch including startup, linker script and source codes, and compile them using arm-none-eabi cross tool chain for arm processors.
I will execute this simple bare metal application on a virtual board by using “qemu” tool.

2 Source code

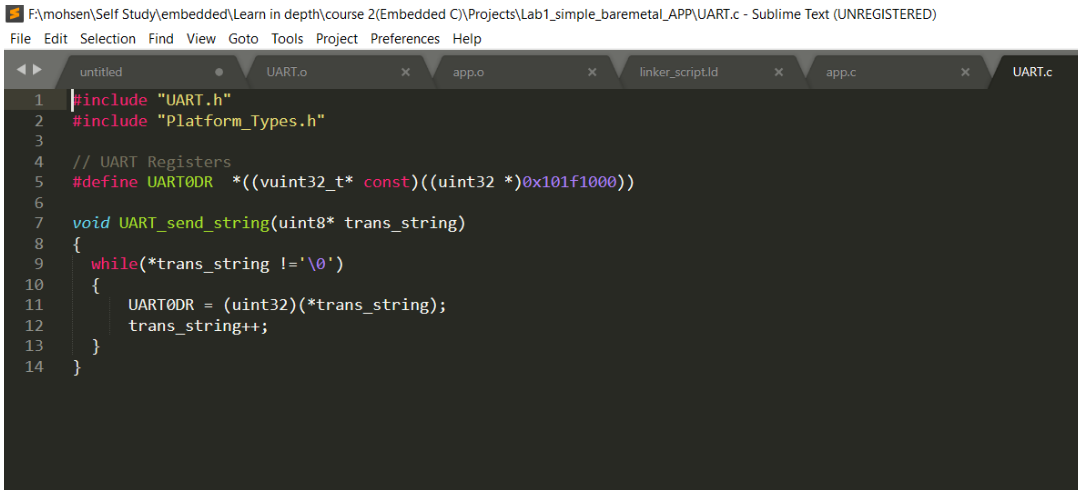
main application code


F:\mohsen\Self Study\embedded\Learn in depth\course 2(Embedded C)\Projects\Lab1_simple_baremetal_APP\app.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
untitled
UART.o
app.o
linker_script.ld
app.c

```c
#include "UART.h"
#include "Platform_Types.h"
sint8 uart_message[]="Learn-in-depth :<mohsen> ";
void main()
{
    UART_send_string(uart_message);
}
```

In the main application code, which I called “app.c” I defined an array of characters and initialized it by the message which will be sent via UART, then pass the message to a function called “UART_send_string” which is defined in included header file “UART.h”.

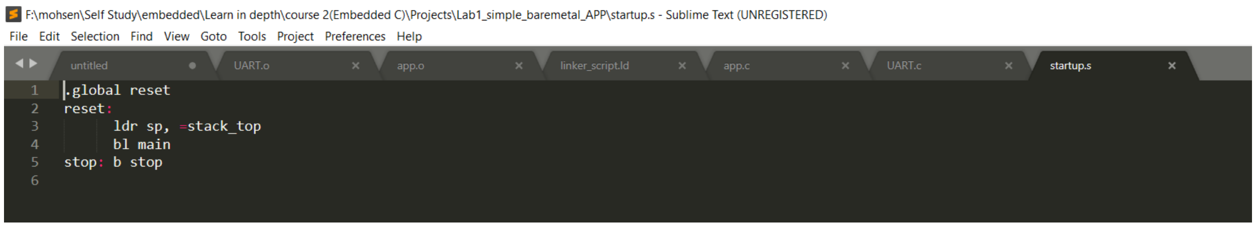
uart code


F:\mohsen\Self Study\embedded\Learn in depth\course 2(Embedded C)\Projects\Lab1_simple_baremetal_APP\UART.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
untitled
UART.o
app.o
linker_script.ld
app.c
UART.c

```c
#include "UART.h"
#include "Platform_Types.h"

// UART Registers
#define UART0DR  *((vuint32_t* const)((uint32 *)0x101f1000))

void UART_send_string(uint8* trans_string)
{
  while(*trans_string !='\0')
  {
     UART0DR = (uint32)(*trans_string);
     trans_string++;
  }
}
```

I will send my message via UART port 0 which is mapped in address 0x101f1000. From specs there is the register UART0DR which is used to transmit and receiving data to UART byte by byte at offset 0x0, so I need to read or write at the beginning of the memory allocated for the UART0. In UART.c there is a macro will be used to write data at the memory address of UART0DR to be transmitted to UART0, in the function “UART_send_string” there is a while loop which checks the end of message and while it is not the end of message then pass byte by byte to the register UART0DR to be transmitted to the UART0.

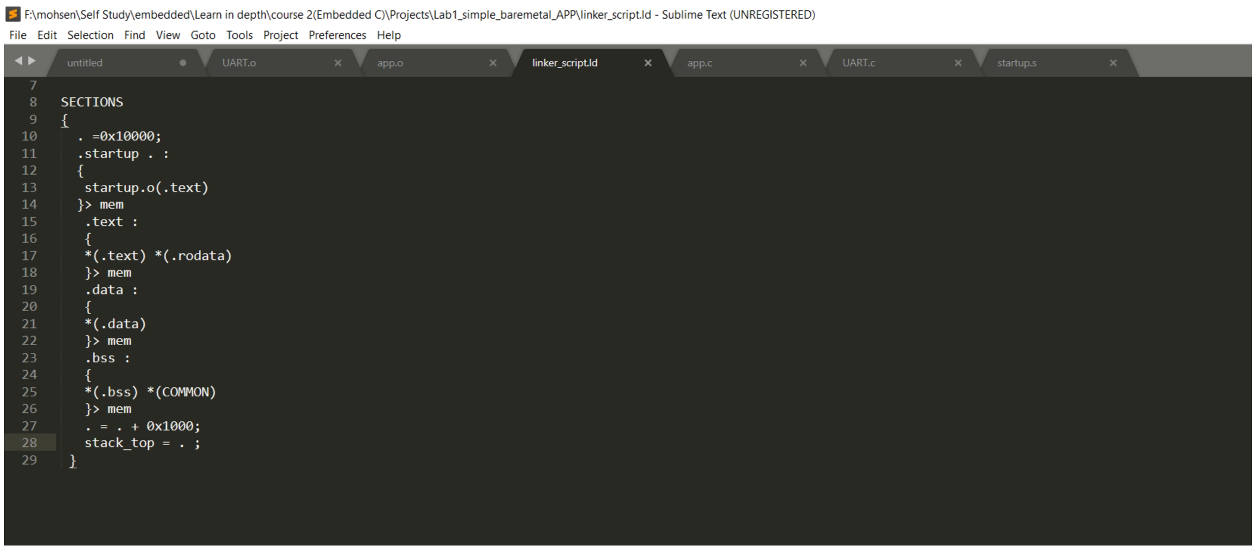
startup code


F:\mohsen\Self Study\embedded\Learn in depth\course 2(Embedded C)\Projects\Lab1_simple_baremetal_APP\startup.s - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
untitled
UART.o
app.o
linker_script.ld
app.c
UART.c
startup.s

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

```

Startup code is written in assembly language because it is dependent on microprocessor architecture, in this simple startup code we just make a reset section which will be first section to execute before main as it will be burned at the processor entry point, in this section I just initialized the stack pointer with “stack_top” which is a symbol will be resolved while linking process from linker script, then just brunch to main function and after that loop in your position and don’t do anything.

linker script


F:\mohsen\Self Study\embedded\Learn in depth\course 2(Embedded C)\Projects\Lab1_simple_baremetal_APP\linker_script.ld - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
untitled
UART.o
app.o
linker_script.ld
app.c
UART.c
startup.s

```

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

In this simple linker script, we define the entry point of my whole application which is the reset section in the startup code. Then we define memory boundaries and its attributes, here I have one memory I called it mem which have attributes (read, write and execute).

The last section in linker script is that divide my whole code in all files to organized sections to be burned to the micro-controller, here I used the location counter
and set it to the entry point of the processor (from specs) 0x10000 and I created a section
will be loaded in this address contains the .text(instructions) of the startup code, after this
section the remained .text of whole other file will be combined to .text section and will be loaded to the memory after .reset section, the last section is ,.data section which combines whole global data of whole files and will be loaded after .text section.

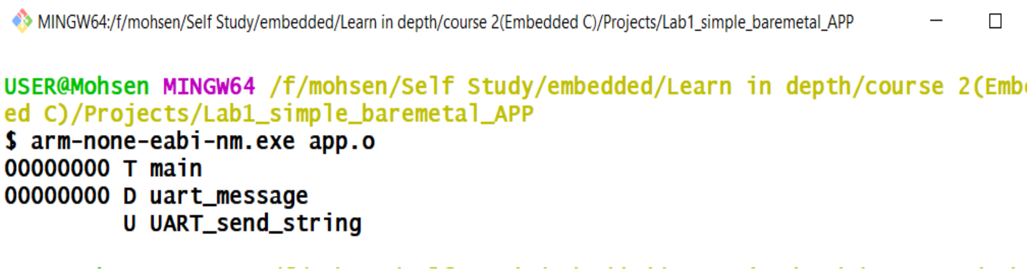
Now the location counter is having the address of the end of .data section so I will add 0x1000 which equals 4MB (my stack size) and make a symbol to be used in startup to initialize stack pointer register.

3 Symbols

Using nm binary utility, I can hack every binary file and see its symbols and which section every symbol belongs to and address of every symbol. In object files there is only
virtual addresses, and every symbol will take a real load address after linking process in the elf image.

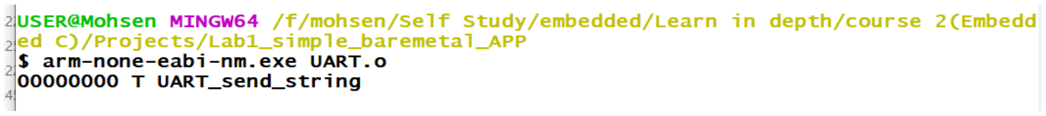
app.o symbols

this object file contains three symbols,
1. Main: which is in text section.
2. UART_send_string: which is unresolved and will be resolved when link
this file to UART.o file.
3. Uart_message: which is in data section.



```
MINGW64:/f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedded C)/Projects/Lab1_simple_baremetal_APP

USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Emb
ed C)/Projects/Lab1_simple_baremetal_APP
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D uart_message
         U UART_send_string
```

uart.o symbols



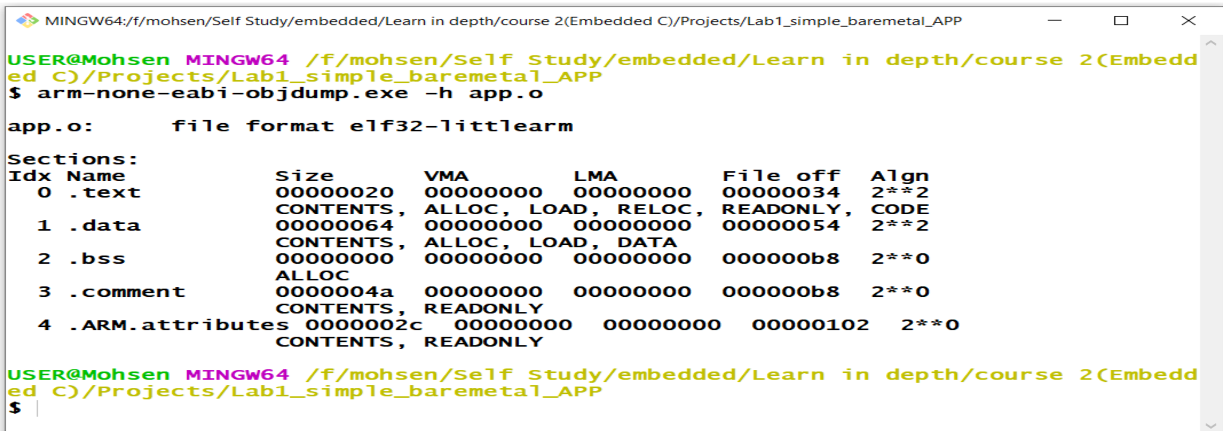
```
USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$ arm-none-eabi-nm.exe UART.o
00000000 T UART_send_string
```

this object file contains only one symbol,
1. Send_uart_message: which is in text section

4 Sections headers

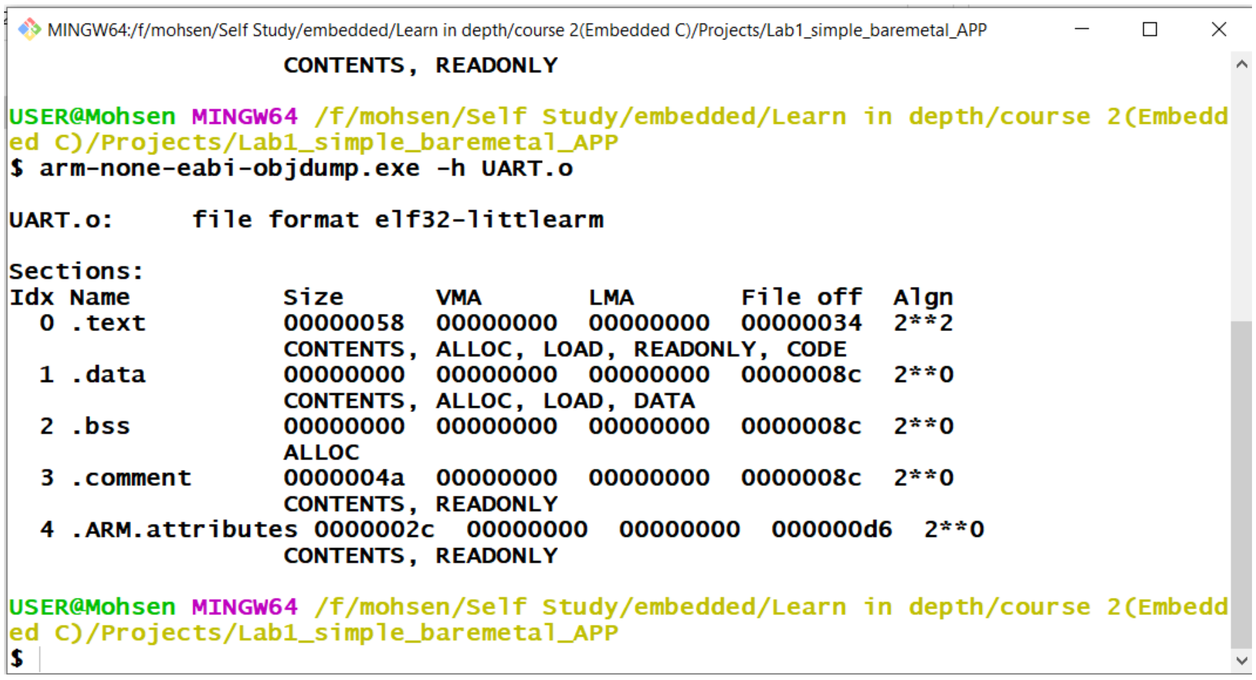
In this section we just care about .text, .data, .bss and .rodata sections, linker may add some other sections like .ARM.attributes and .comment but we don’t care about these sections as it will be excluded in the final executable file and wont be loaded the micro-controller.

app.o sections header



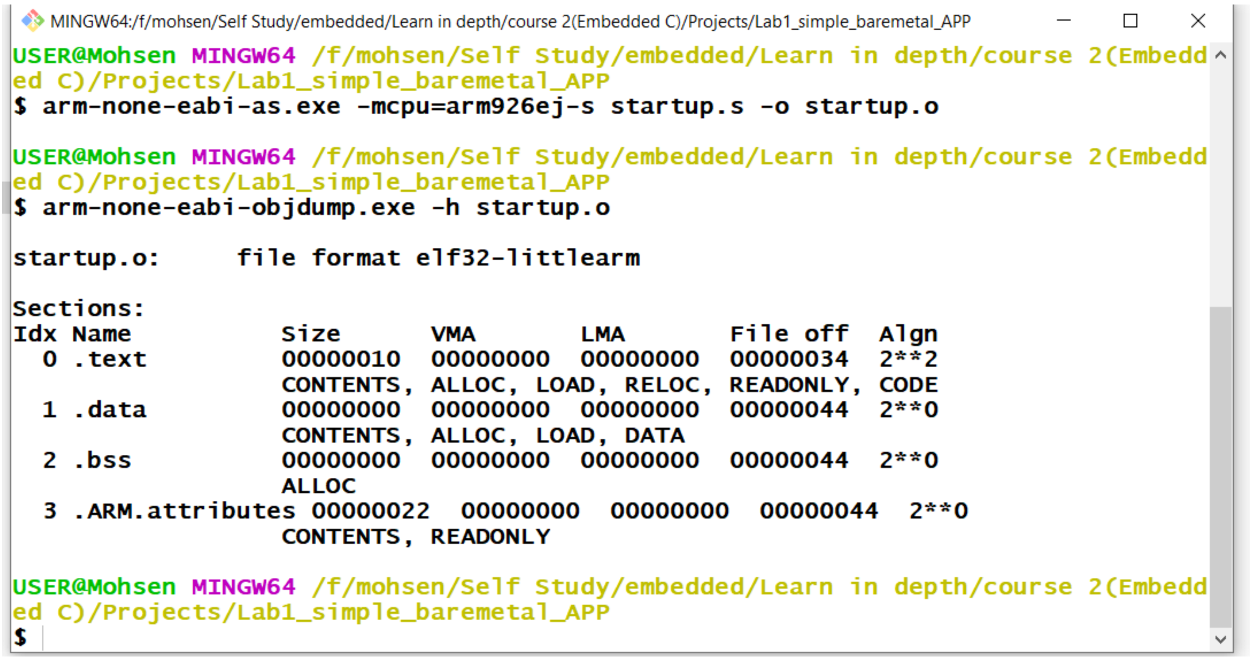
```
MINGW64:/f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedded C)/Projects/Lab1_simple_baremetal_APP

USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000020  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000064  00000000  00000000  00000054  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b8  2**0
                  ALLOC
  3 .comment      0000004a  00000000  00000000  000000b8  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 0000002c  00000000  00000000  00000102  2**0
                  CONTENTS, READONLY

USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$
```

uart.o sections



```
MINGW64:/f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedded C)/Projects/Lab1_simple_baremetal_APP
                  CONTENTS, READONLY

USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$ arm-none-eabi-objdump.exe -h UART.o

UART.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000058  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  0000008c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  0000008c  2**0
                  ALLOC
  3 .comment      0000004a  00000000  00000000  0000008c  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 0000002c  00000000  00000000  000000d6  2**0
                  CONTENTS, READONLY

USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$
```

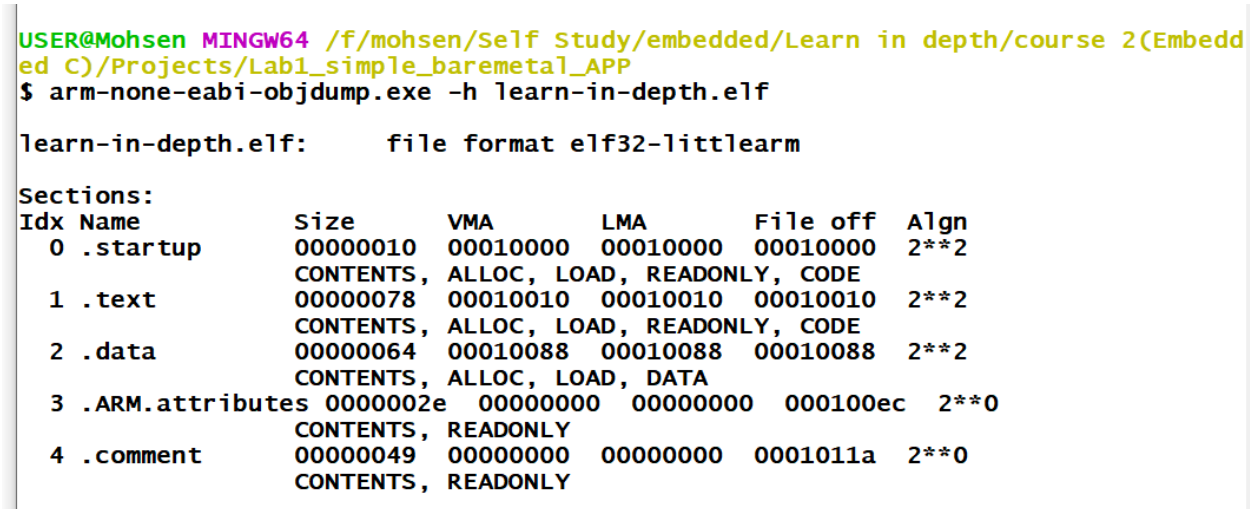
startup.o sections



MINGW64:/f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedded C)/Projects/Lab1_simple_baremetal_APP

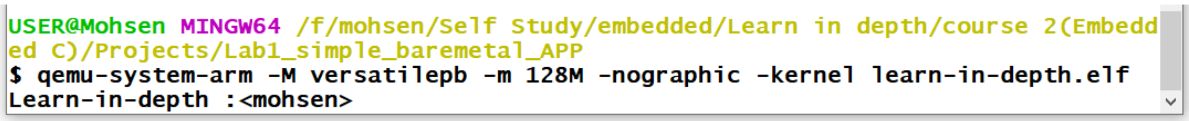
```
USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$ arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o

USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$ arm-none-eabi-objdump.exe -h startup.o

startup.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000010  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000044  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000044  2**0
                  ALLOC
  3 .ARM.attributes 00000022  00000000  00000000  00000044  2**0
                  CONTENTS, READONLY

USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$
```

final elf image sections



```
USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$ arm-none-eabi-objdump.exe -h learn-in-depth.elf

learn-in-depth.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .startup      00000010  00010000  00010000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         00000078  00010010  00010010  00010010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         00000064  00010088  00010088  00010088  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  000100ec  2**0
                  CONTENTS, READONLY
  4 .comment      00000049  00000000  00000000  0001011a  2**0
                  CONTENTS, READONLY
```

5 executing application using qemu tool



```
USER@Mohsen MINGW64 /f/mohsen/Self Study/embedded/Learn in depth/course 2(Embedd
ed C)/Projects/Lab1_simple_baremetal_APP
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn-in-depth.elf
Learn-in-depth :<mohsen>
```