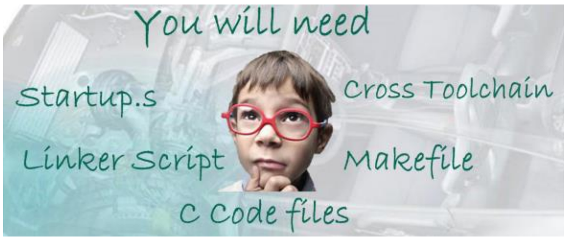
Lab1(Embedded C lesson2)



Requirements

In this lab we want to send a string through the UART of ARM VersatilePB board

Cross toolchain

Before we start we should know what we need to complete this task

- We get the cross toolchain by installing qemu and arm toolchain
- the next step we want to impelement the c code files

C code files

uart.c

```c
#include "uart.h"
// define uart register to send data
#define UART0DR *(volatile unsigned int* const)((unsigned int*)0x101f1000)

void Uart_Send_String(unsigned char* P_tx_string)
{
    while(*P_tx_string != '\0')
    {
        UART0DR = (unsigned int)(*P_tx_string);
        P_tx_string++; //next char ...
    }
}
```

We prepare the UART0DR to send the string on it

uart.h

```c
#ifndef UART_H_
#define UART_H_

void Uart_Send_String(unsigned char* P_tx_string);

#endif
```

Define the function in uart.c file

app.c

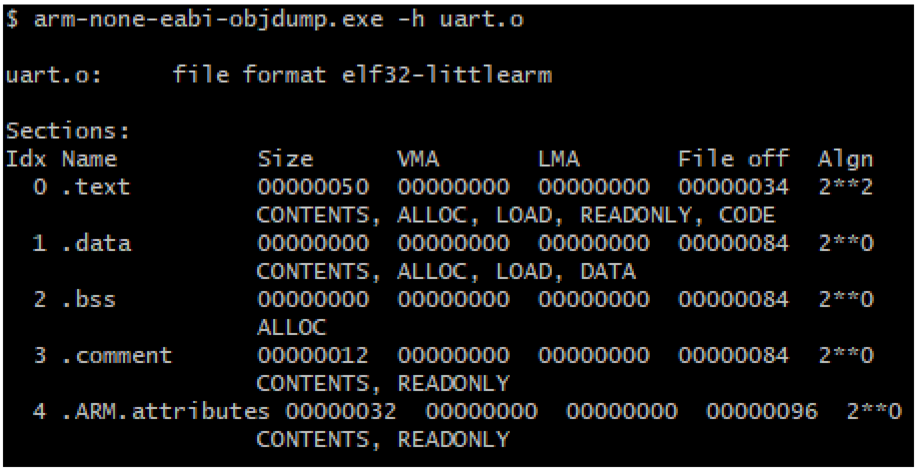
```c
#include "uart.h"

unsigned char string_buffer[100] = "Learn-in-depth:<AliTaima>";
unsigned char const string_buffer2[100] = "to create a rodate section";
void main(void)
{
    // Send on physical board
    Uart_Send_String(string_buffer);
}
```

Prepare the string that we want to send through the uart

Object files

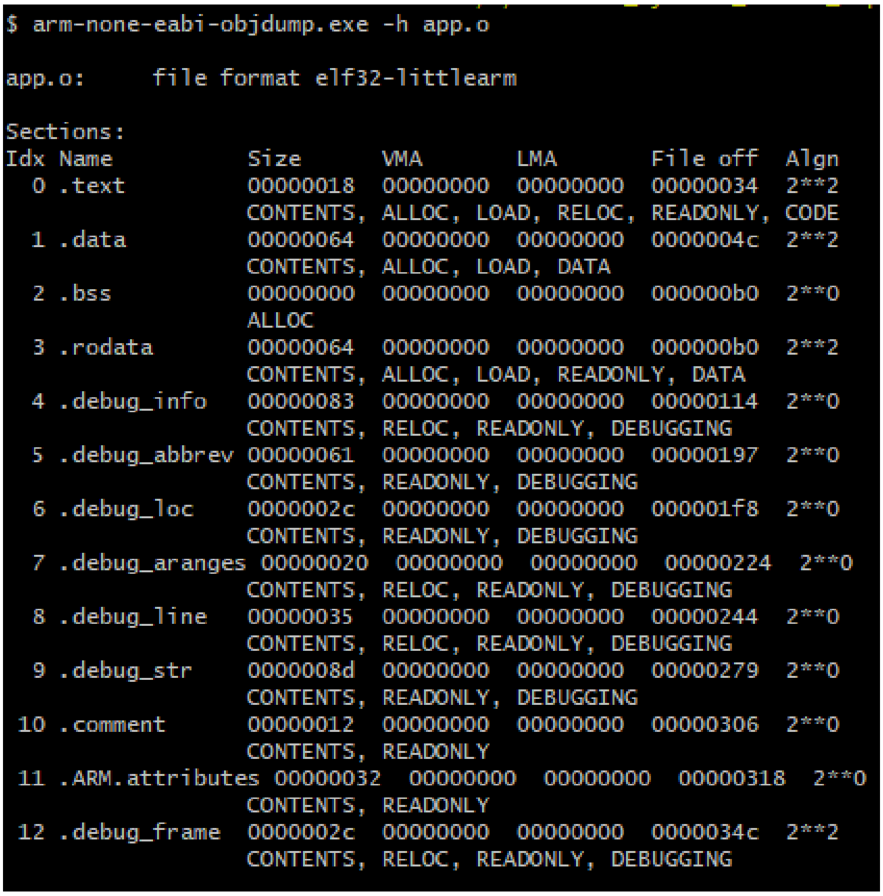
We can get the object files of c files by using arm toolchain

arm-none-eabi-gcc.exe -c -I . -mcpu=arm926ej-s uart.c -o uart.o → `here we won't to get the debug section`



```
$ 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         00000050  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000084  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000084  2**0
                  ALLOC
  3 .comment      00000012  00000000  00000000  00000084  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000032  00000000  00000000  00000096  2**0
                  CONTENTS, READONLY
```

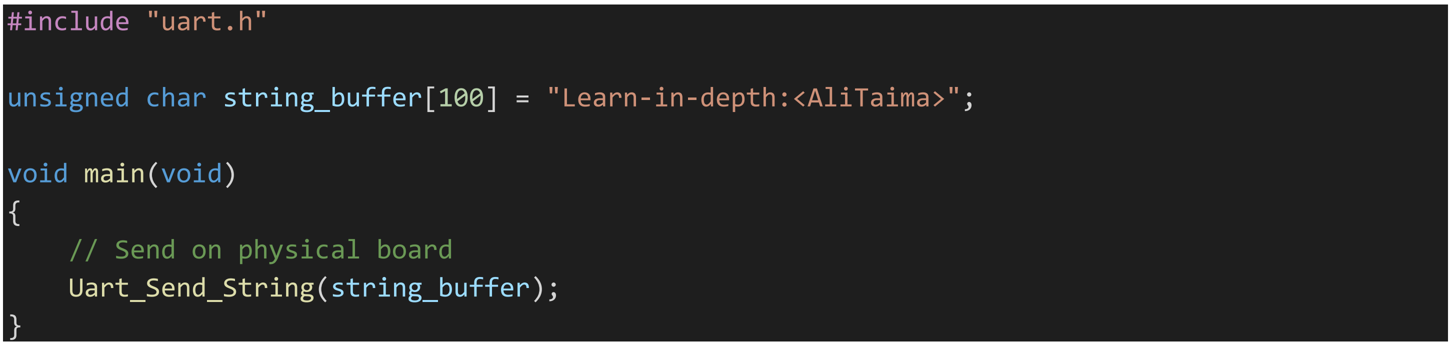
arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s app.c -o app.o → `here we will get debug sections of app file`



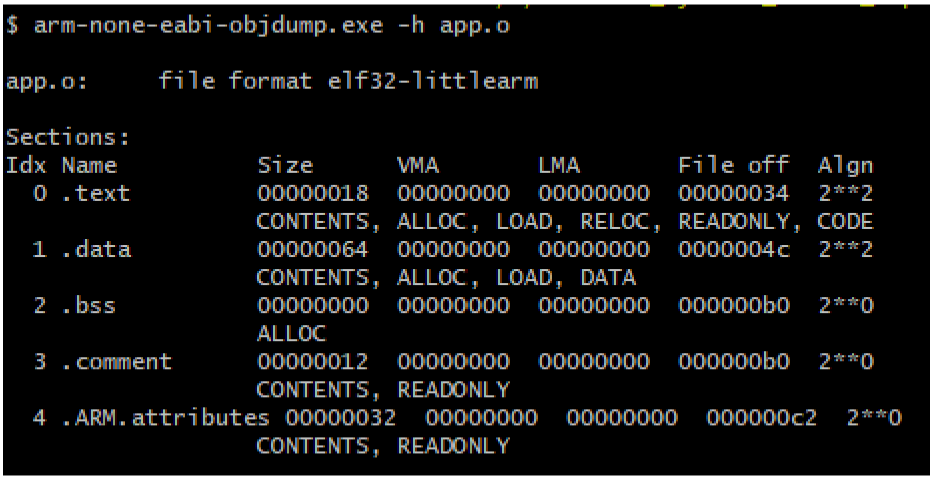
```
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000018  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000064  00000000  00000000  0000004c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b0  2**0
                  ALLOC
  3 .rodata       00000064  00000000  00000000  000000b0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .debug_info   00000083  00000000  00000000  00000114  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  5 .debug_abbrev 00000061  00000000  00000000  00000197  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_loc    0000002c  00000000  00000000  000001f8  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_aranges 00000020  00000000  00000000  00000224  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_line   00000035  00000000  00000000  00000244  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  9 .debug_str    0000008d  00000000  00000000  00000279  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .comment      00000012  00000000  00000000  00000306  2**0
                  CONTENTS, READONLY
 11 .ARM.attributes 00000032  00000000  00000000  00000318  2**0
                  CONTENTS, READONLY
 12 .debug_frame  0000002c  00000000  00000000  0000034c  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

- We can see that there is a .rodata section in app.c
    - The reason for this we add a constant in our code but if we delete it we won't get the .rodata section

After adding removing the constant we can’t see the **.rodata** section



```c
#include "uart.h"

unsigned char string_buffer[100] = "Learn-in-depth:<AliTaima>";

void main(void)
{
    // Send on physical board
    Uart_Send_String(string_buffer);
}
```



```
$ 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         00000018  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000064  00000000  00000000  0000004c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b0  2**0
                  ALLOC
  3 .comment      00000012  00000000  00000000  000000b0  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000032  00000000  00000000  000000c2  2**0
                  CONTENTS, READONLY
```

Until now we have the object file of our c files(app.o, uart.o)

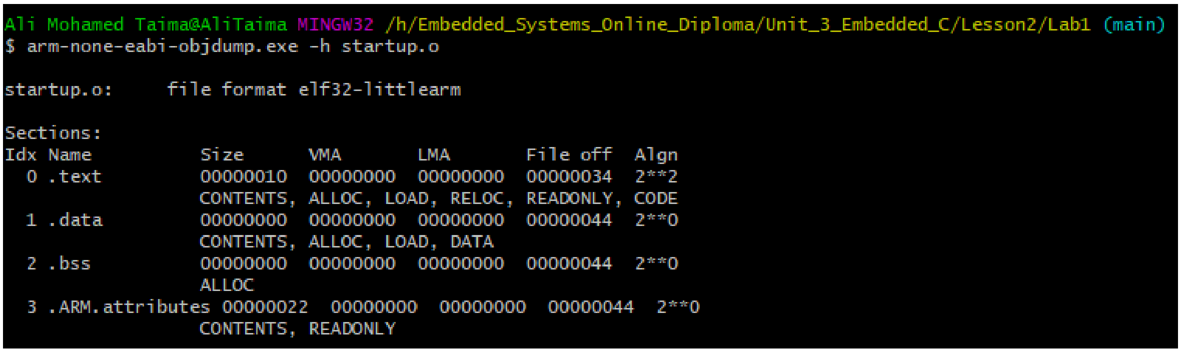
Startup.s

Now we want to make the startup.s to be the startup code of our program

```
@make the reset visible from other files

.global reset

reset:

        ldr sp, =stack_top

        bl main

stop: b stop
```

- In this file we set the stack pointer, and make the startup function is main function
    - In this step we can change the main function to any other name to start with
- We want to get the object file of our startup file
    - `arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o`

We can show the sctions in the startup.o file



```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ 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
```

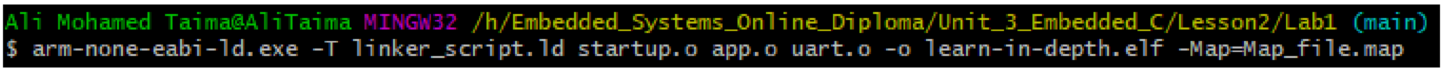
Linker script

```
ENTRY(reset)

MEMORY

{

        Mem (rwx) : ORIGIN = 0x00000000, LENGTH = 64M

}

SECTIONS

{

        . = 0x10000;

        .startup . :

        {

                startup.o(.text)

        }> Mem

        .text :

        {

                *(.txt) *(.rodata)

        }> Mem

        .data :

        {

                *(.data)

        }> Mem

        .bss :

        {

                *(.bss) *(COMMON)

        }> Mem

        . = . + 0x1000; /* 4KB of stack Memory */

        stack_top = .;

}
```

In this file we use only one memory and set our stack_top address

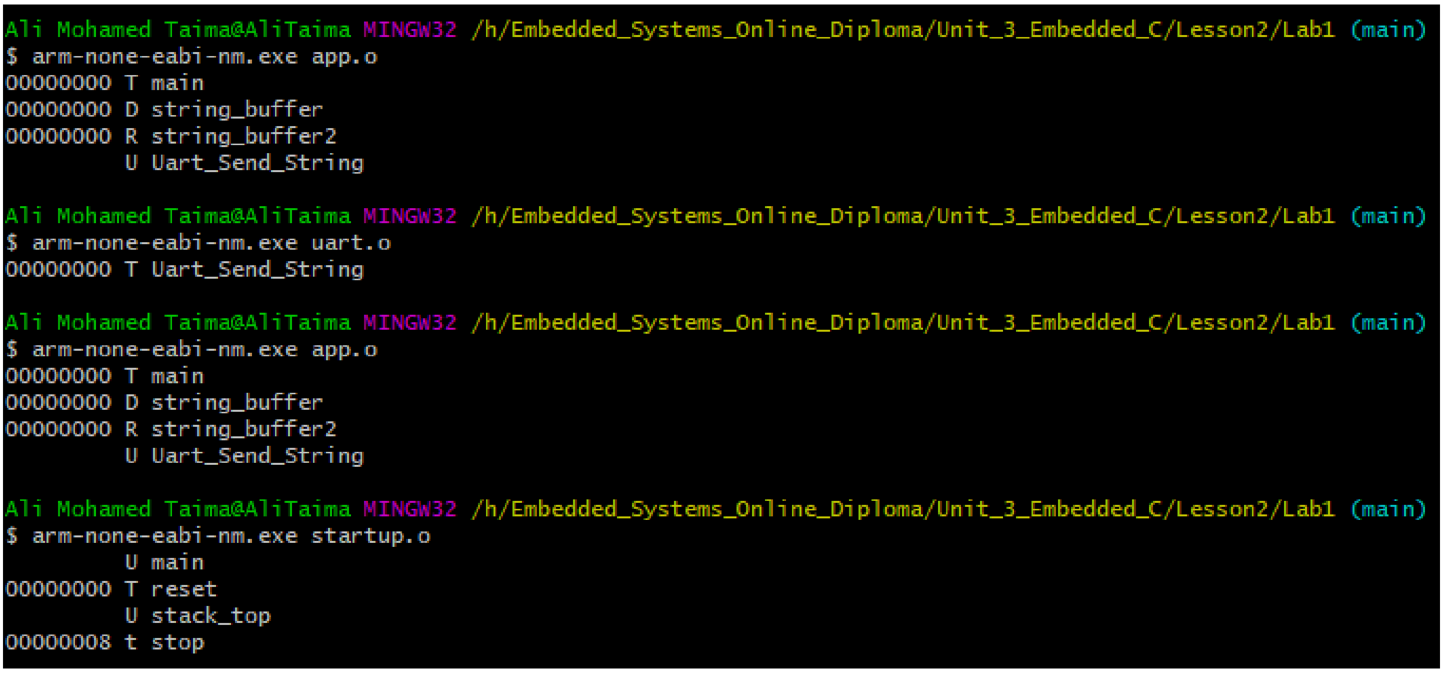
Eng. Ali Taima@11/2022

We want to link our files



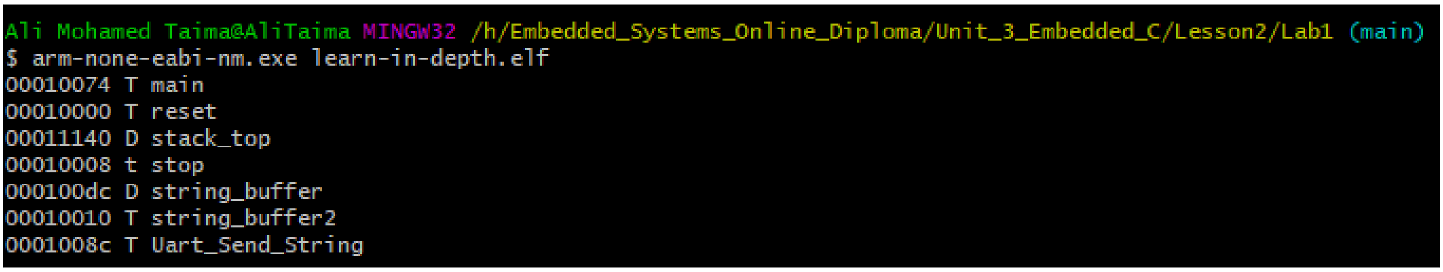
```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-ld.exe -T linker_script.ld startup.o app.o uart.o -o learn-in-depth.elf -Map=Map_file.map
```

Symbols of files

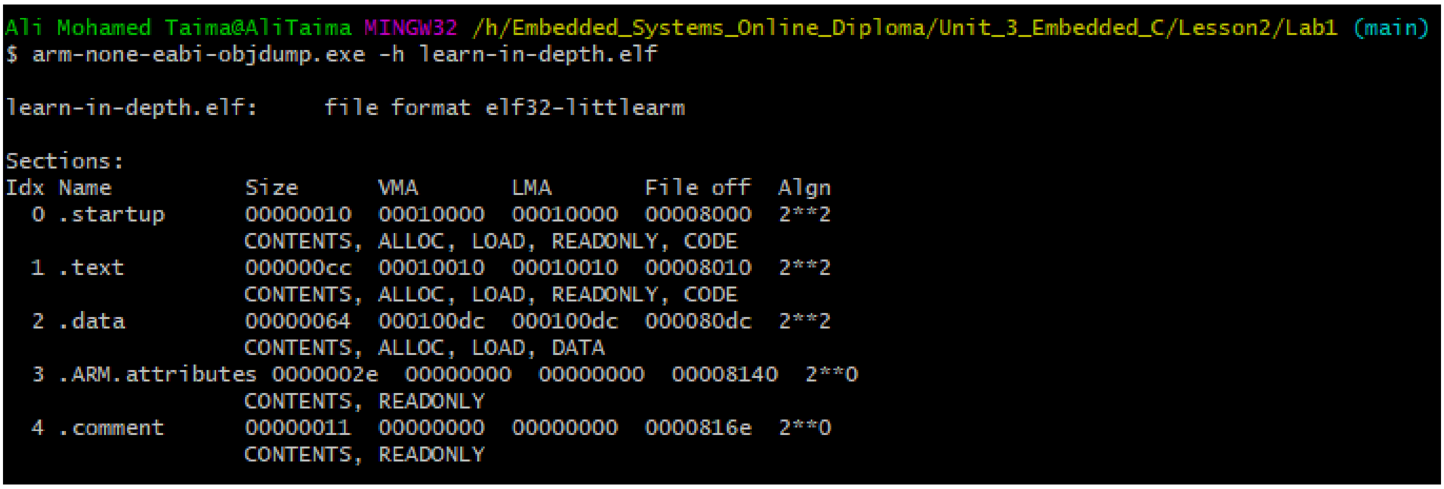


```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_buffer
00000000 R string_buffer2
         U Uart_Send_String

Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-nm.exe uart.o
00000000 T Uart_Send_String

Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_buffer
00000000 R string_buffer2
         U Uart_Send_String

Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-nm.exe startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop
```

We combine all symbols in learn-in-depth.elf file as we show



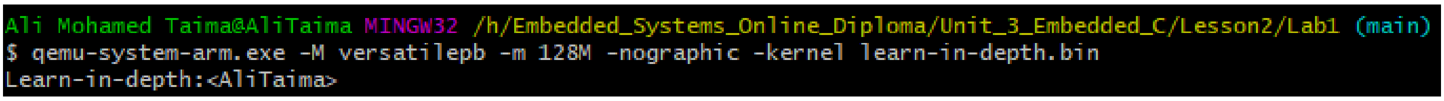
```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-nm.exe learn-in-depth.elf
00010074 T main
00010000 T reset
00011140 D stack_top
00010008 t stop
000100dc D string_buffer
00010010 T string_buffer2
0001008c T Uart_Send_String
```



```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ 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  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         000000cc  00010010  00010010  00008010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         00000064  000100dc  000100dc  000080dc  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  00008140  2**0
                  CONTENTS, READONLY
  4 .comment      00000011  00000000  00000000  0000816e  2**0
                  CONTENTS, READONLY
```

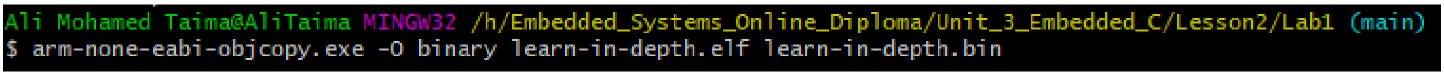
Alo se can show that all sections combined in learn-in-depth file

Now we want to get the binary file to burn in our MC



```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ qemu-system-arm.exe -M versatilepb -m 128M -nographic -kernel learn-in-depth.bin
Learn-in-depth:<AliTaima>
```

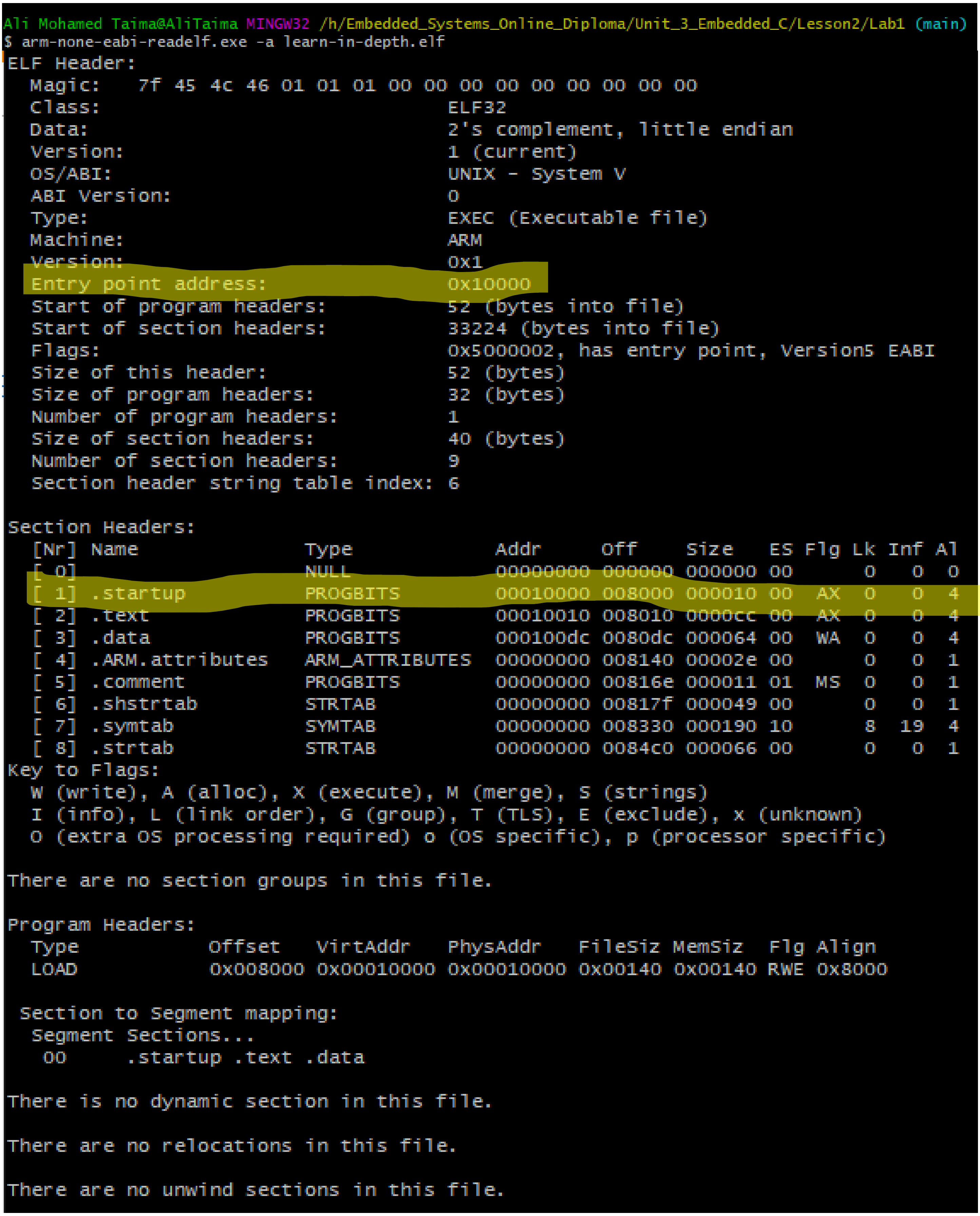
Finaly we can use the qemu emulator to show our string output



```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin
```

Eng. Ali Taima@11/2022

We can make sure from our entry point by using the **readelf Binary utilities** command



```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson2/Lab1 (main)
$ arm-none-eabi-readelf.exe -a learn-in-depth.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x10000
  Start of program headers:          52 (bytes into file)
  Start of section headers:          33224 (bytes into file)
  Flags:                             0x5000002, has entry point, Version5 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         1
  Size of section headers:           40 (bytes)
  Number of section headers:         9
  Section header string table index: 6

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .startup          PROGBITS        00010000 008000 000010 00  AX  0   0  4
  [ 2] .text             PROGBITS        00010010 008010 0000cc 00  AX  0   0  4
  [ 3] .data             PROGBITS        000100dc 0080dc 000064 00  WA  0   0  4
  [ 4] .ARM.attributes   ARM_ATTRIBUTES  00000000 008140 00002e 00      0   0  1
  [ 5] .comment          PROGBITS        00000000 00816e 000011 01  MS  0   0  1
  [ 6] .shstrtab         STRTAB          00000000 00817f 000049 00      0   0  1
  [ 7] .symtab           SYMTAB          00000000 008330 000190 10      8  19  4
  [ 8] .strtab           STRTAB          00000000 0084c0 000066 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x008000 0x00010000 0x00010000 0x00140 0x00140 RWE 0x8000

 Section to Segment mapping:
  Segment Sections...
   00     .startup .text .data

There is no dynamic section in this file.

There are no relocations in this file.

There are no unwind sections in this file.
```