Lab 1

- In this lab we will write c code, startup code and linker script and use all binary utilities such as objdump ,nm , objcopy and readelf
- We will write code from scratch and send string to uart0 and uart0 will display it on board name : versatilepb(arm926ej-s).
- We will write the whole code and execute it with using arm none eabi tools without any IDE.
- Note: in this lab I already put the paths of arm and qemu in system variables in my computer

From specs we found :

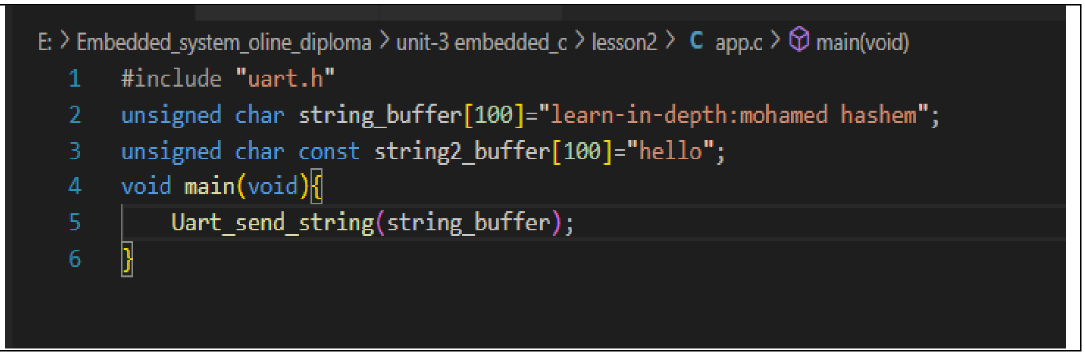
Entry point of processor is : 0x10000

To activate UART0 you just write on UART0DR register (32bit).

And its address is :0x101f1000

Codes :

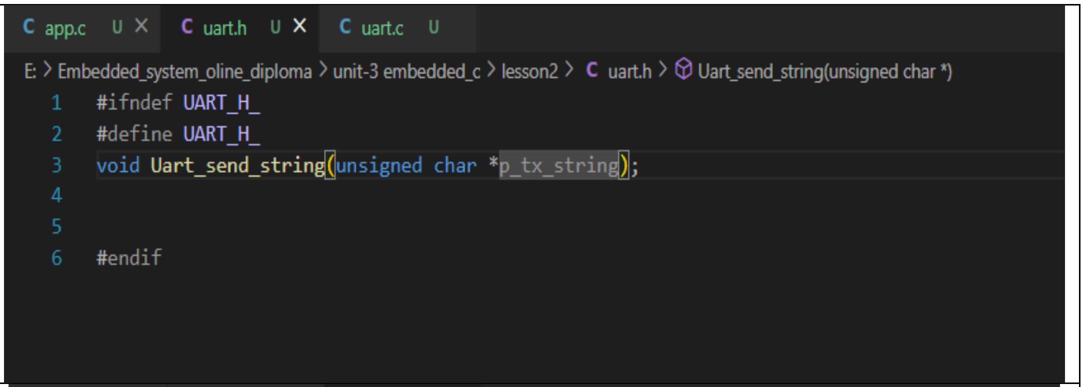
app.c



E: > Embedded_system_oline_diploma > unit-3 embedded_c > lesson2 > C app.c > main(void)

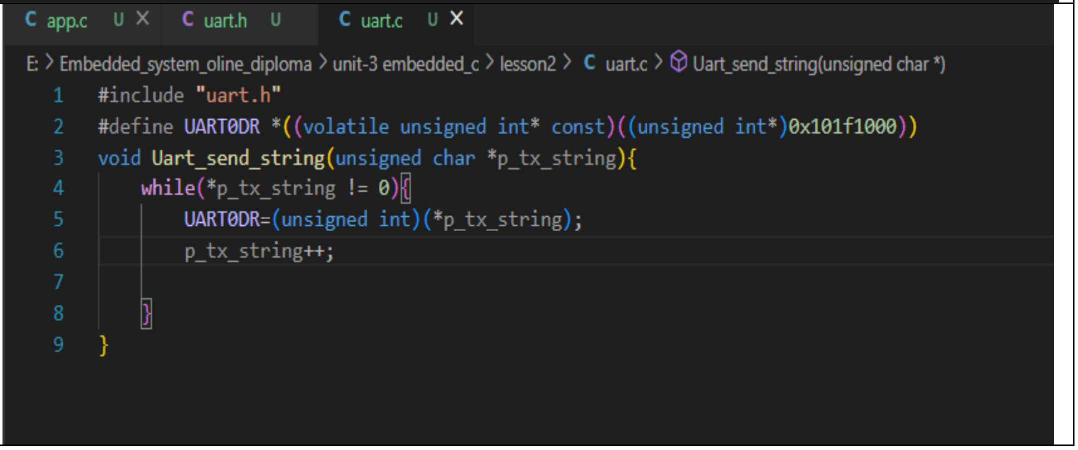
```c
#include "uart.h"
unsigned char string_buffer[100]="learn-in-depth:mohamed hashem";
unsigned char const string2_buffer[100]="hello";
void main(void){
    Uart_send_string(string_buffer);
}
```

uart.h


C app.c U ×
C uart.h U ×
C uart.c U
E: > Embedded_system_oline_diploma > unit-3 embedded_c > lesson2 > C uart.h > Uart_send_string(unsigned char *)

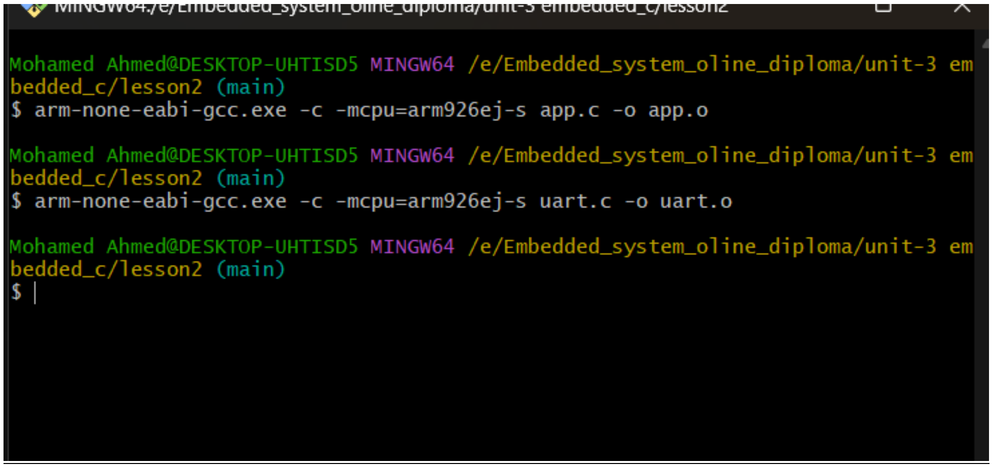
```c
#ifndef UART_H_
#define UART_H_
void Uart_send_string(unsigned char *p_tx_string);


#endif
```

uart.c


C app.c U ×
C uart.h U
C uart.c U ×
E: > Embedded_system_oline_diploma > unit-3 embedded_c > lesson2 > C uart.c > Uart_send_string(unsigned char *)

```c
#include "uart.h"
#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++;

    }
}
```

Let’s open git bash and use arm toolchain to get app.o and uart.o :

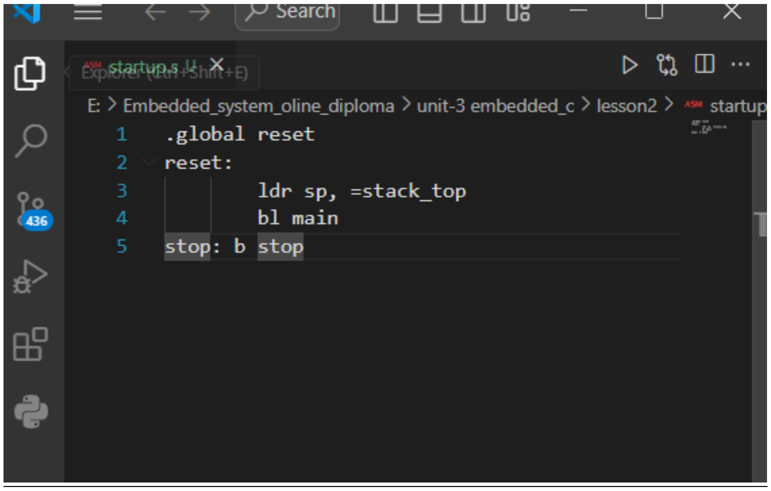

MINGW64:/e/Embedded_system_oline_diploma/unit-3 embedded_c/lesson2

```
Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s app.c -o app.o

Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s uart.c -o uart.o

Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ |
```

Now let’s write our startup…

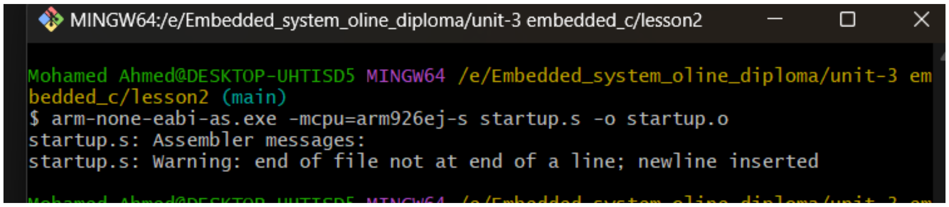
Startup.s :

- we defined reset as global so we can access it when we write our linker script


Search
startup.s U
Explorer (Ctrl+Shift+E)
E: > Embedded_system_oline_diploma > unit-3 embedded_c > lesson2 > startup
436

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

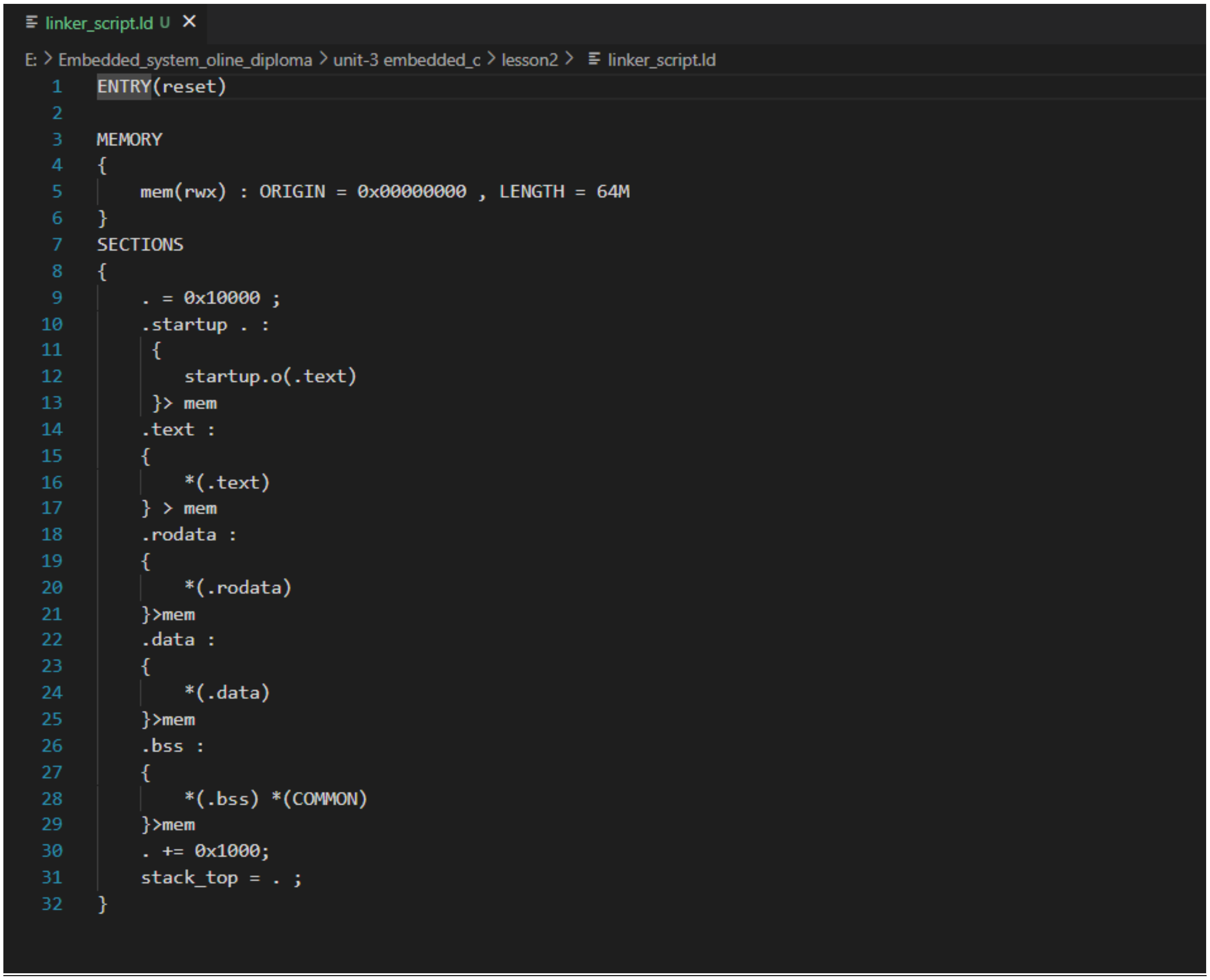
Then we generate startup.o using assembler :


MINGW64:/e/Embedded_system_oline_diploma/unit-3 embedded_c/lesson2

```
Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted

Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
```

Linker script :

- in linker script we control all memory locations, memory sizes and starting point of our program and stack size.

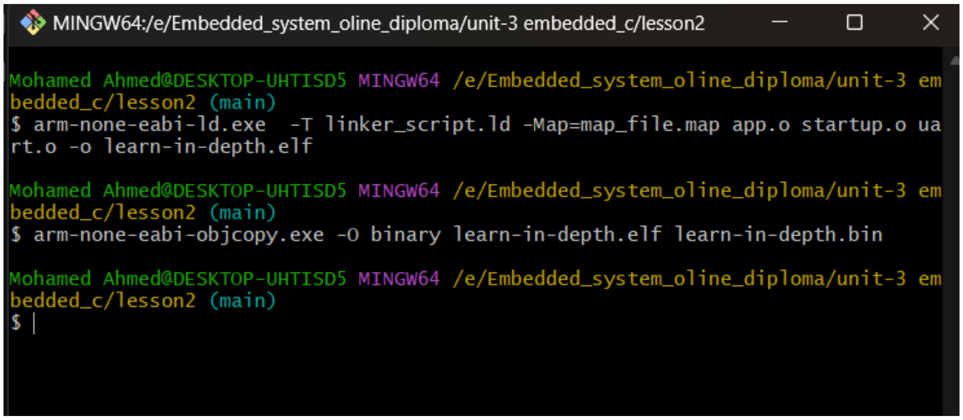


linker_script.ld U ×
E: > Embedded_system_oline_diploma > unit-3 embedded_c > lesson2 > linker_script.ld

```
ENTRY(reset)

MEMORY
{
    mem(rwx) : ORIGIN = 0x00000000 , LENGTH = 64M
}
SECTIONS
{
    . = 0x10000 ;
    .startup . :
     {
        startup.o(.text)
     }> mem
    .text :
    {
        *(.text)
    } > mem
    .rodata :
    {
        *(.rodata)
    }>mem
    .data :
    {
        *(.data)
    }>mem
    .bss :
    {
        *(.bss) *(COMMON)
    }>mem
    . += 0x1000;
    stack_top = . ;
}
```

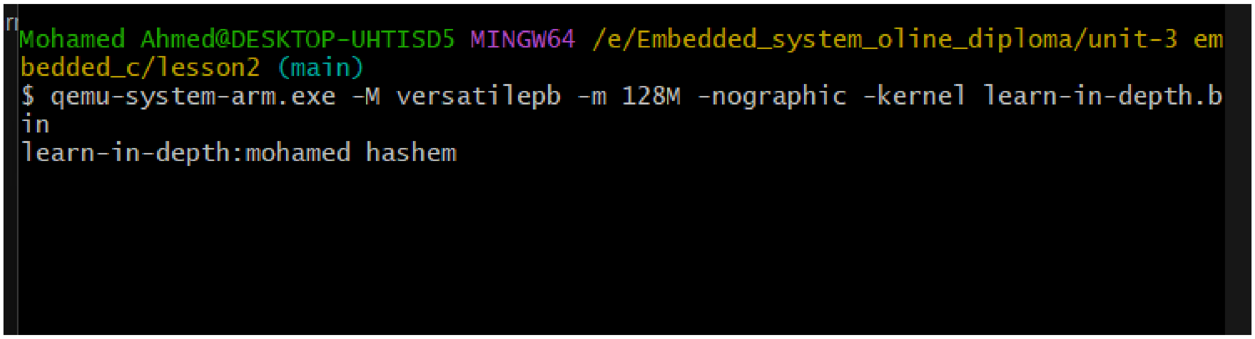
Then we will link all object files app.o, startup.o and uart.o with linker script using linker and generate our .elf file and .map file .

Then generate binary code that will be burnt on board.


MINGW64:/e/Embedded_system_oline_diploma/unit-3 embedded_c/lesson2

```
Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ arm-none-eabi-ld.exe  -T linker_script.ld -Map=map_file.map app.o startup.o ua
rt.o -o learn-in-depth.elf

Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin

Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ |
```

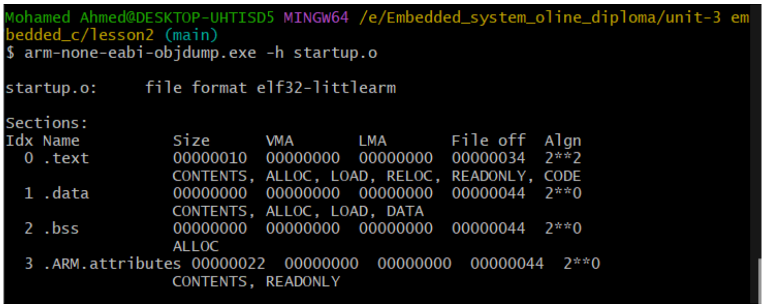
Now we will call qemo emulator to run the code on the board and see the expected output : learn-in-depth: mohamed hashem



```
Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ qemu-system-arm.exe -M versatilepb -m 128M -nographic -kernel learn-in-depth.b
in
learn-in-depth:mohamed hashem
```

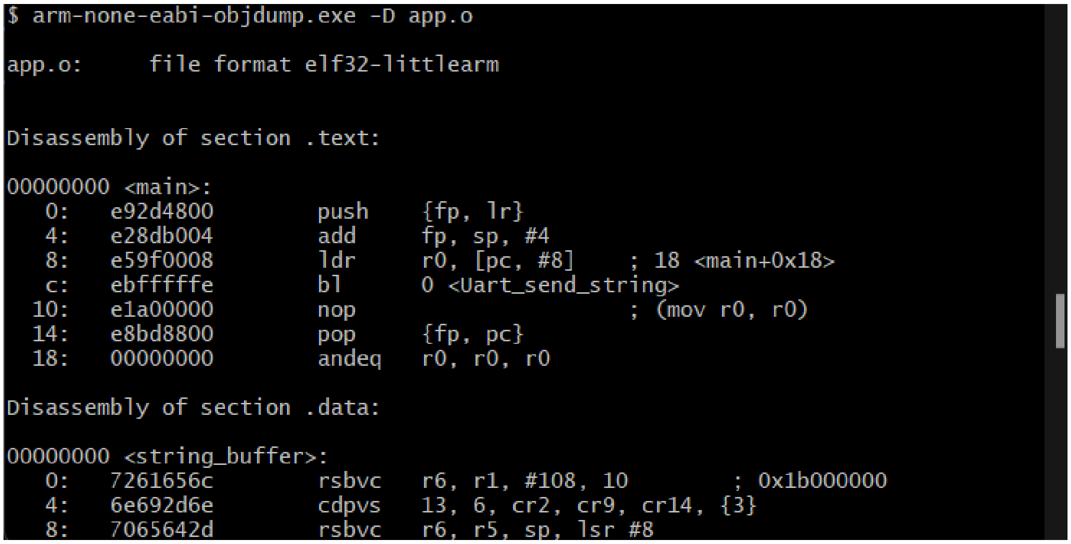
Lets use some binary utilities to differentiate between different stages of code :

- objdump -h to show header sections



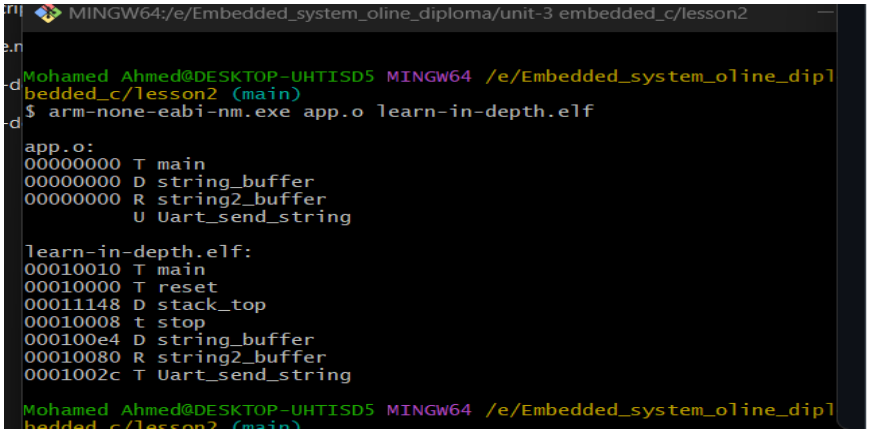
```
Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_diploma/unit-3 em
bedded_c/lesson2 (main)
$ arm-none-eabi-objdump.exe -h startup.o

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
```

- objdump -D to dissemble the code in details



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

app.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0:   e92d4800        push    {fp, lr}
   4:   e28db004        add     fp, sp, #4
   8:   e59f0008        ldr     r0, [pc, #8]    ; 18 <main+0x18>
   c:   ebfffffe        bl      0 <Uart_send_string>
  10:   e1a00000        nop                     ; (mov r0, r0)
  14:   e8bd8800        pop     {fp, pc}
  18:   00000000        andeq   r0, r0, r0

Disassembly of section .data:

00000000 <string_buffer>:
   0:   7261656c        rsbvc   r6, r1, #108, 10        ; 0x1b000000
   4:   6e692d6e        cdpvs   13, 6, cr2, cr9, cr14, {3}
   8:   7065642d        rsbvc   r6, r5, sp, lsr #8
```

- nm to show symbols with it’s address



MINGW64:/e/Embedded_system_oline_diploma/unit-3 embedded_c/lesson2

```
Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_dipl
bedded_c/lesson2 (main)
$ arm-none-eabi-nm.exe app.o learn-in-depth.elf

app.o:
00000000 T main
00000000 D string_buffer
00000000 R string2_buffer
         U Uart_send_string

learn-in-depth.elf:
00010010 T main
00010000 T reset
00011148 D stack_top
00010008 t stop
000100e4 D string_buffer
00010080 R string2_buffer
0001002c T Uart_send_string

Mohamed Ahmed@DESKTOP-UHTISD5 MINGW64 /e/Embedded_system_oline_dipl
bedded_c/lesson2 (main)
```