Embedded C Lab 1

Arm 926EJ-S Core

By \ Eng . Belal Hani Abu Sabha

In this lab, I will write bare-metal software to send the string "My Name" using UART on an Arm926EJ-S core.

After reading the datasheet:

This board has four UARTs (UART0 to UART3) , I will use UART0 that has a base address of 0x101f1000. The UART Data Register (UART0DR) is located at an offset of 0x0 from this base address, making physical address 0x101f1000. The entry point for software is address 0x10000 .

Summary of what the software does :

1-When power is applied, the program counter (PC) points to the entry point at address 0x10000. The startup code initializes the hardware , Then code executes the .main() function .

2-In main(), a global variable containing the .string "My Name" is defined .

3-The Uart_Send_String() function is called .with a pointer to this string .

4-In the uart.c file, a pointer is defined to point to the UART0 Data Register address (0x101f1000) .

5-Inside the Uart_Send_String() function, a local pointer is created to point to the input .string .

.

6- A while loop is used to transmit each character. For each character :

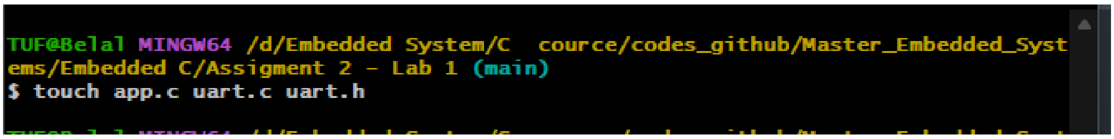
A-The byte is written to UART0DR Register .

B-The pointer is incremented to point to the next character.

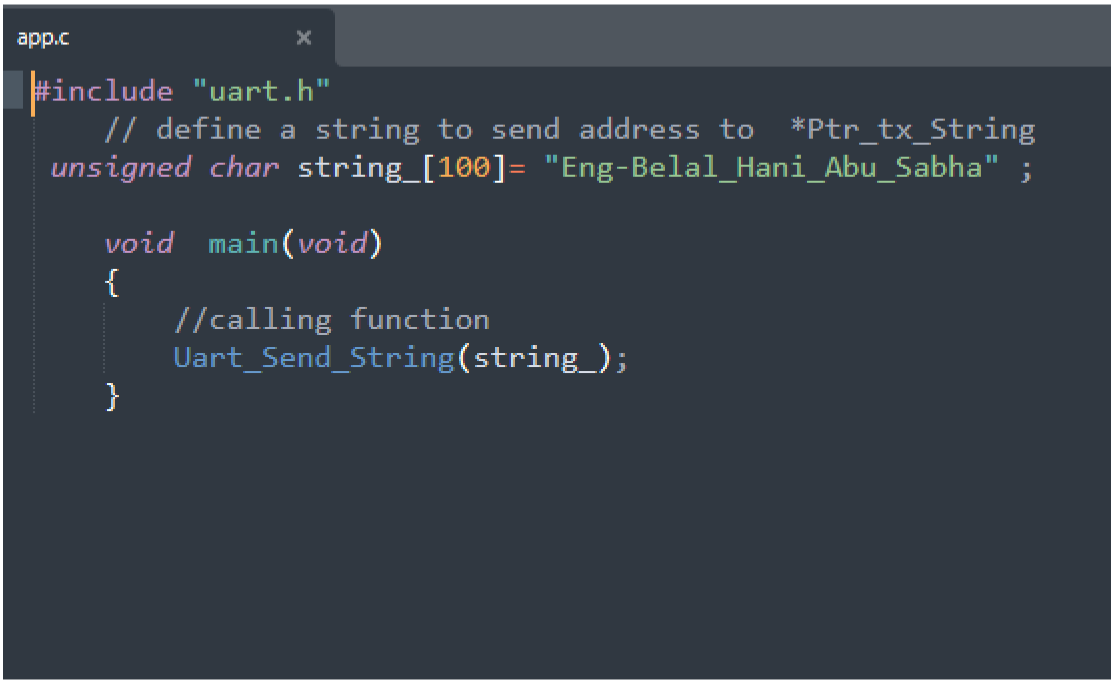
C-The loop continues until the null terminator ('\0') is reached .

Let us Write The Software :

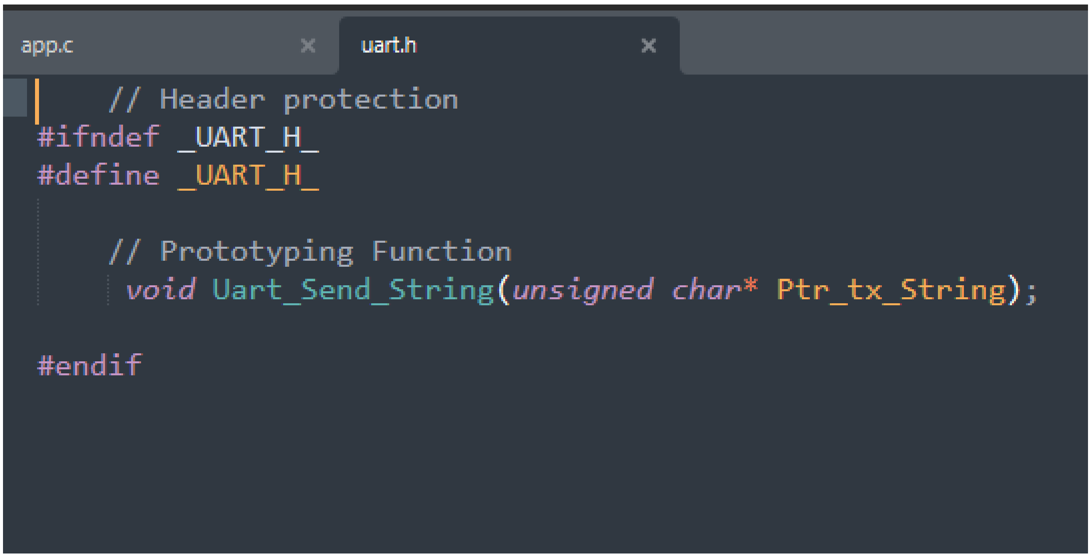
1- We will create three files : app.c , uart.c and uart .h and then write the Code .



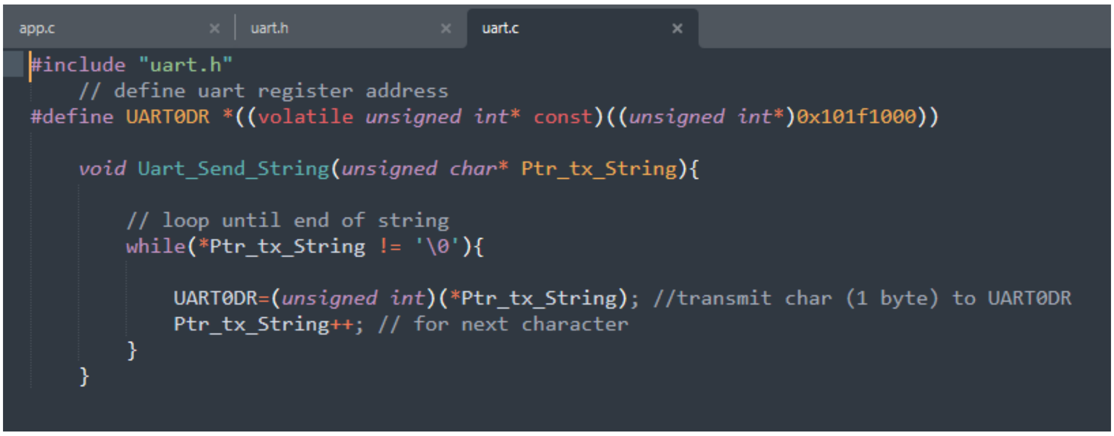
```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - Lab 1 (main)
$ touch app.c uart.c uart.h
```



```
#include "uart.h"
    // define a string to send address to  *Ptr_tx_String
 unsigned char string_[100]= "Eng-Belal_Hani_Abu_Sabha" ;

    void  main(void)
    {
        //calling function
        Uart_Send_String(string_);
    }
```

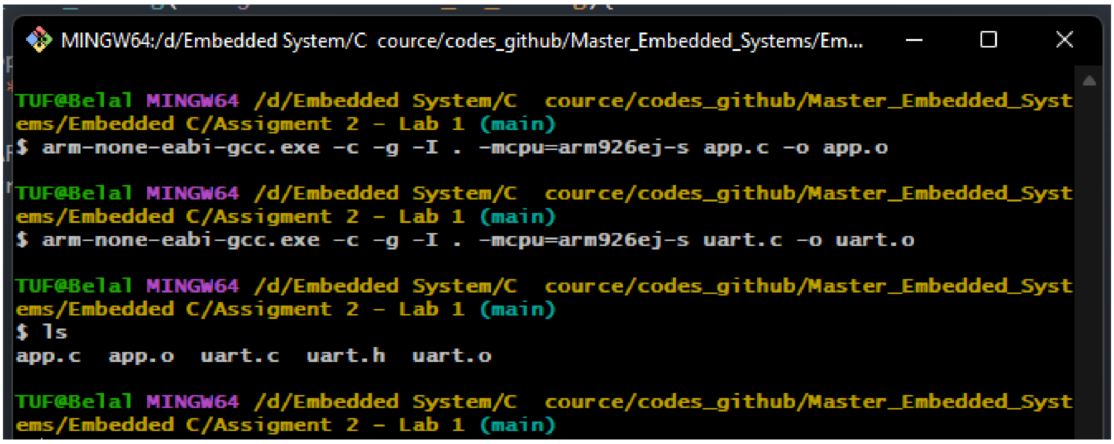


```c
    // Header protection
#ifndef _UART_H_
#define _UART_H_

    // Prototyping Function
     void Uart_Send_String(unsigned char* Ptr_tx_String);

#endif
```

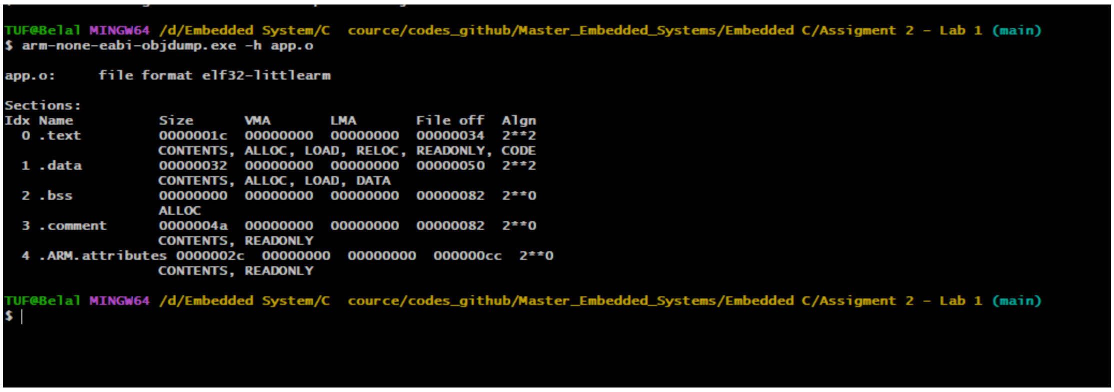


```c
#include "uart.h"
    // define uart register address
#define UART0DR *((volatile unsigned int* const)((unsigned int*)0x101f1000))

    void Uart_Send_String(unsigned char* Ptr_tx_String){

        // loop until end of string
        while(*Ptr_tx_String != '\0'){

            UART0DR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UART0DR
            Ptr_tx_String++; // for next character
        }
    }
```

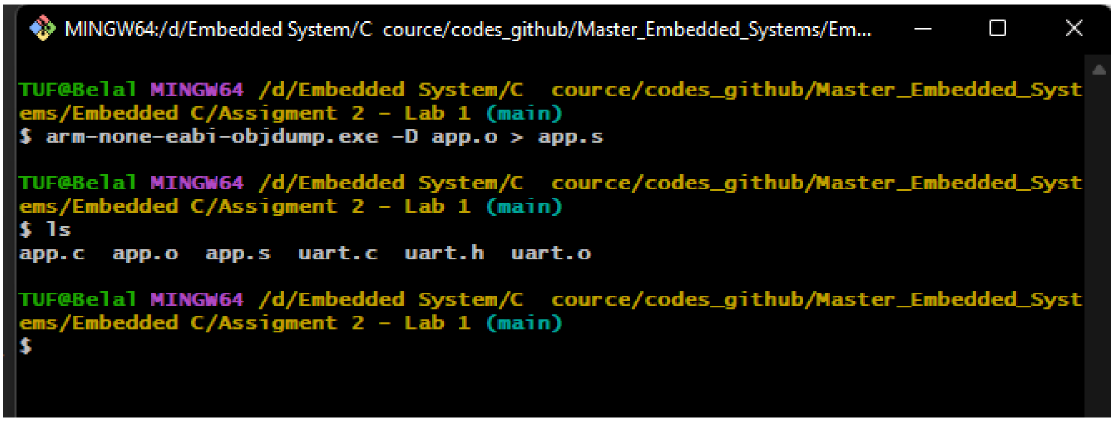
2- We will compile uart.c , app.c to generate app.o , uart.o then show sections from app.c and generate an assembly file for app.c


MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Em...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s app.c -o app.o

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s uart.c -o uart.o

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ ls
app.c   app.o   uart.c   uart.h   uart.o

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
```

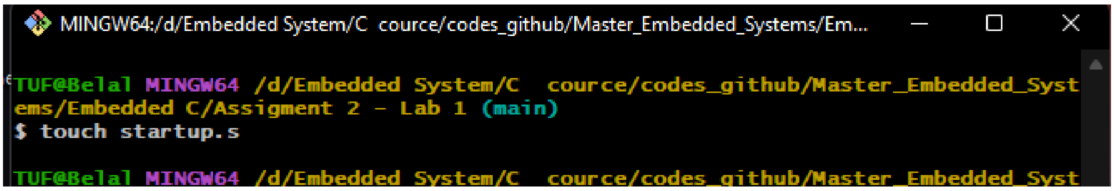


```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - Lab 1 (main)
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000001c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000032  00000000  00000000  00000050  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000082  2**0
                  ALLOC
  3 .comment      0000004a  00000000  00000000  00000082  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 0000002c  00000000  00000000  000000cc  2**0
                  CONTENTS, READONLY

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - Lab 1 (main)
$ |
```

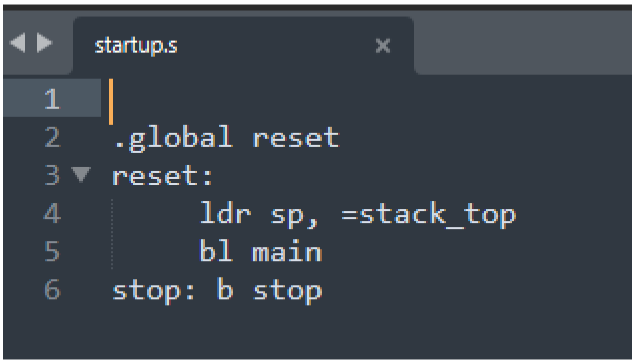

MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Em...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ arm-none-eabi-objdump.exe -D app.o > app.s

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ ls
app.c   app.o   app.s   uart.c   uart.h   uart.o

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$
```

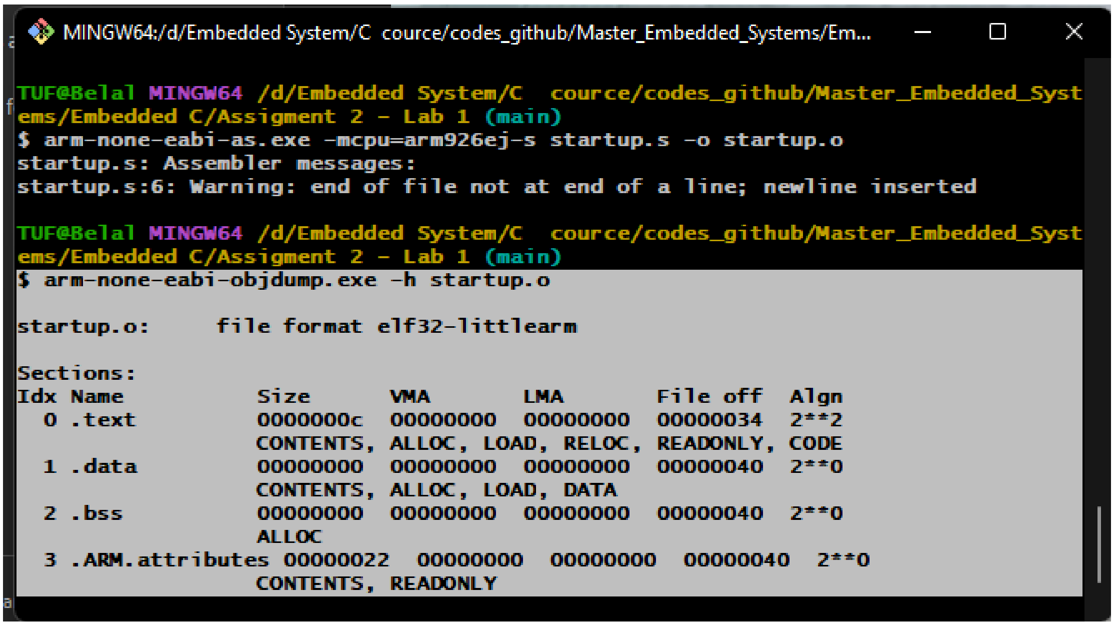
3- Create startup.s , compile it then display its sections .


MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Em...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ touch startup.s

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
```

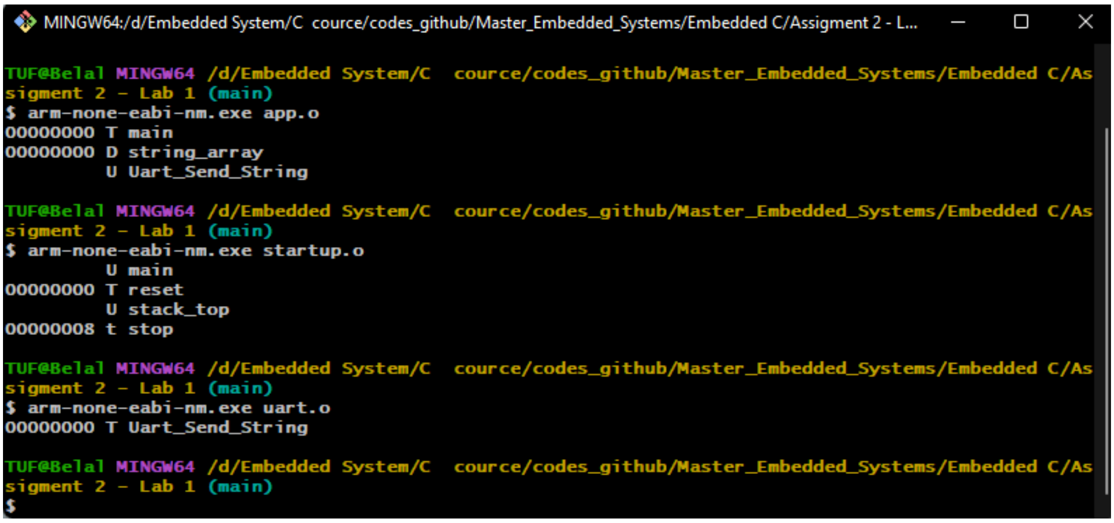


```

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


MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Em...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o
startup.s: Assembler messages:
startup.s:6: Warning: end of file not at end of a line; newline inserted

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (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         0000000c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000040  2**0
                  ALLOC
  3 .ARM.attributes 00000022  00000000  00000000  00000040  2**0
                  CONTENTS, READONLY
```

4-When we display the symbols we note that addresses of the symbols are virtual (not physical addresses)

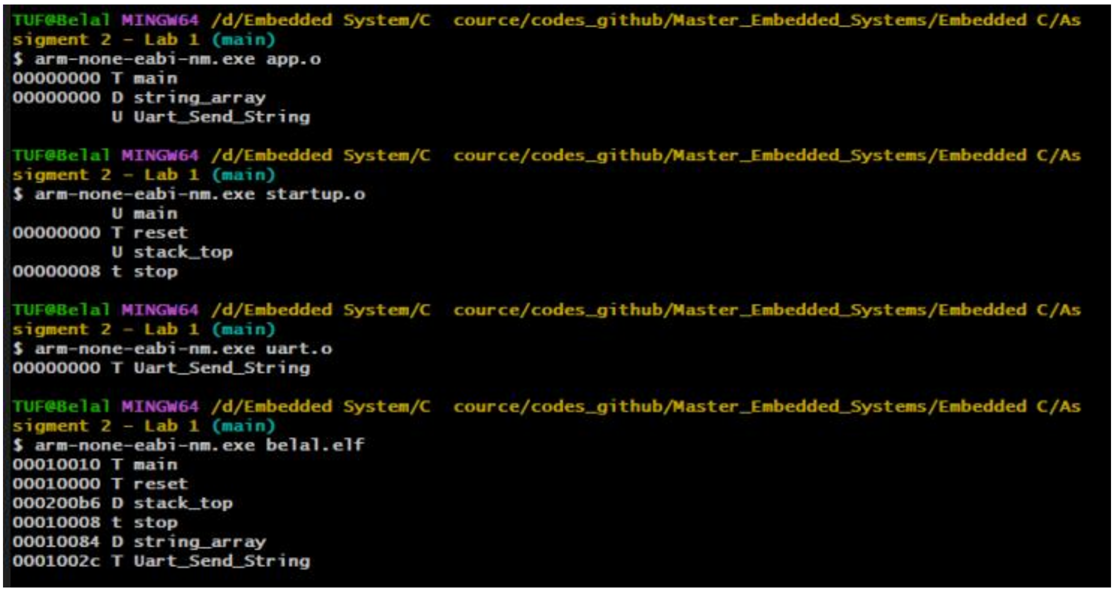


MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - L...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_array
         U Uart_Send_String

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-nm.exe startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-nm.exe uart.o
00000000 T Uart_Send_String

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$
```

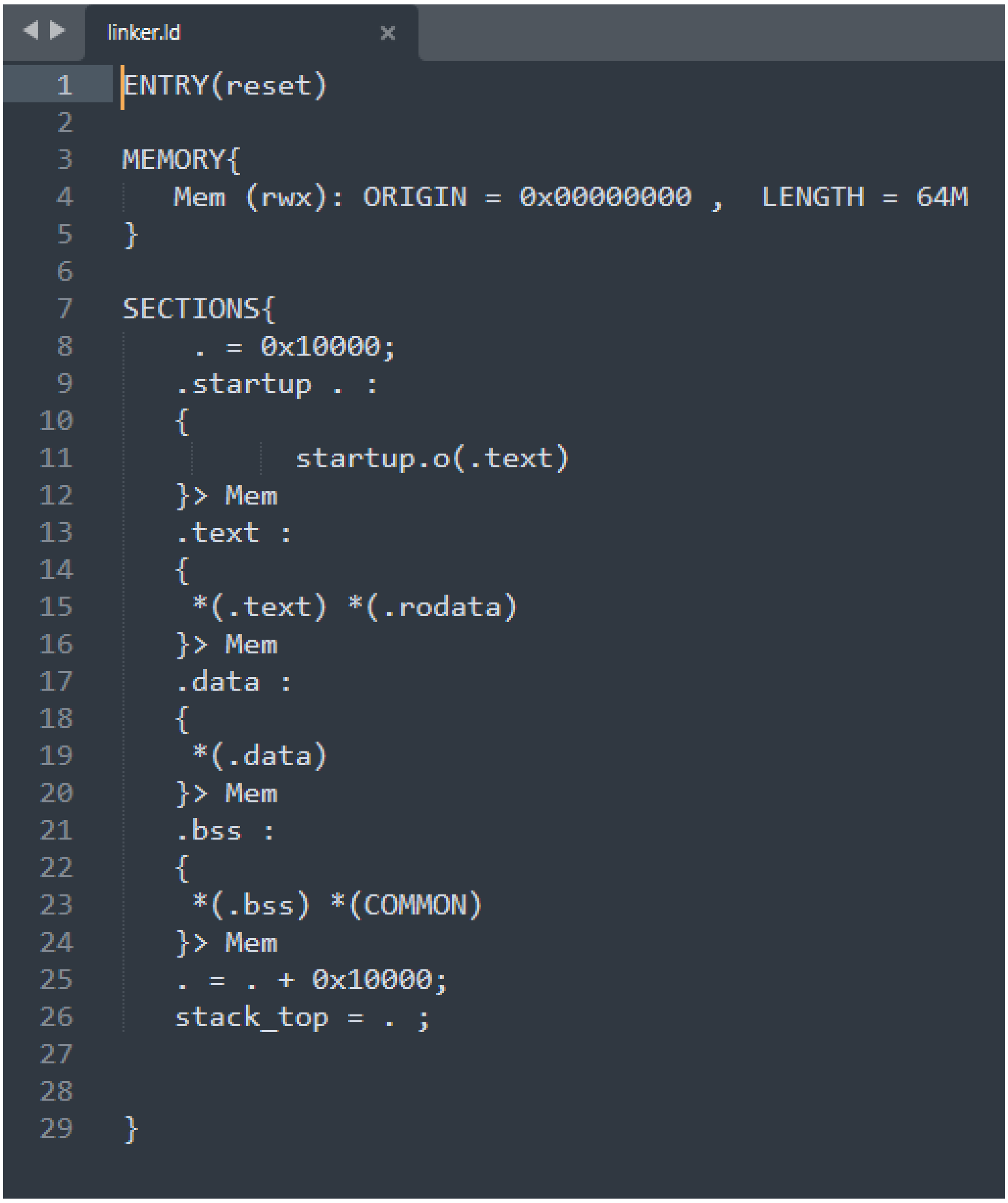
So we need to write a linker script to assign a physical address to every section then link all .o files to generate .elf file .

After linking:

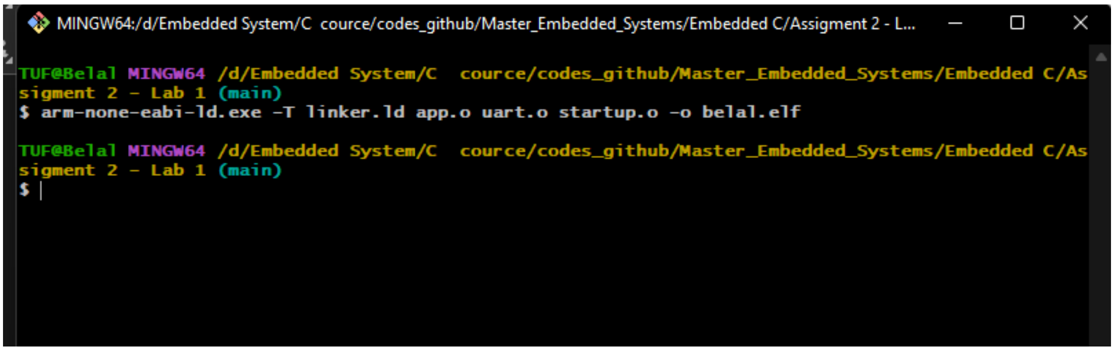


```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_array
         U Uart_Send_String

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-nm.exe startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-nm.exe uart.o
00000000 T Uart_Send_String

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-nm.exe belal.elf
00010010 T main
00010000 T reset
000200b6 D stack_top
00010008 t stop
00010084 D string_array
0001002c T Uart_Send_String
```

Linker Script :

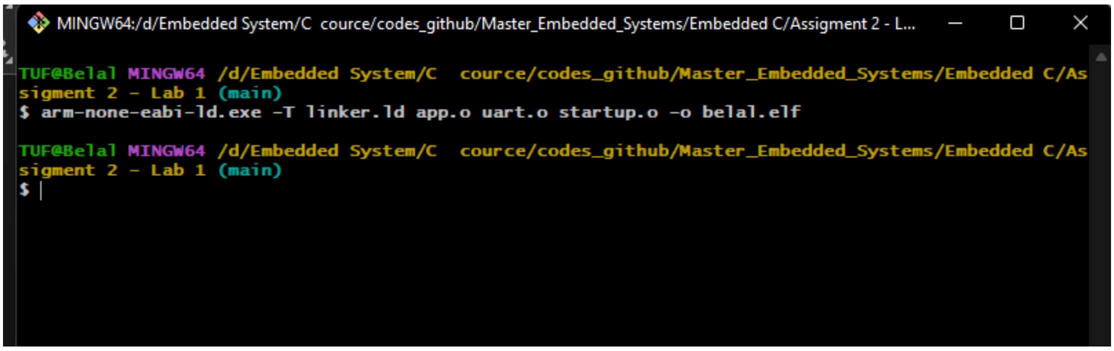


```
ENTRY(reset)

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

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


}
```


MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - L...

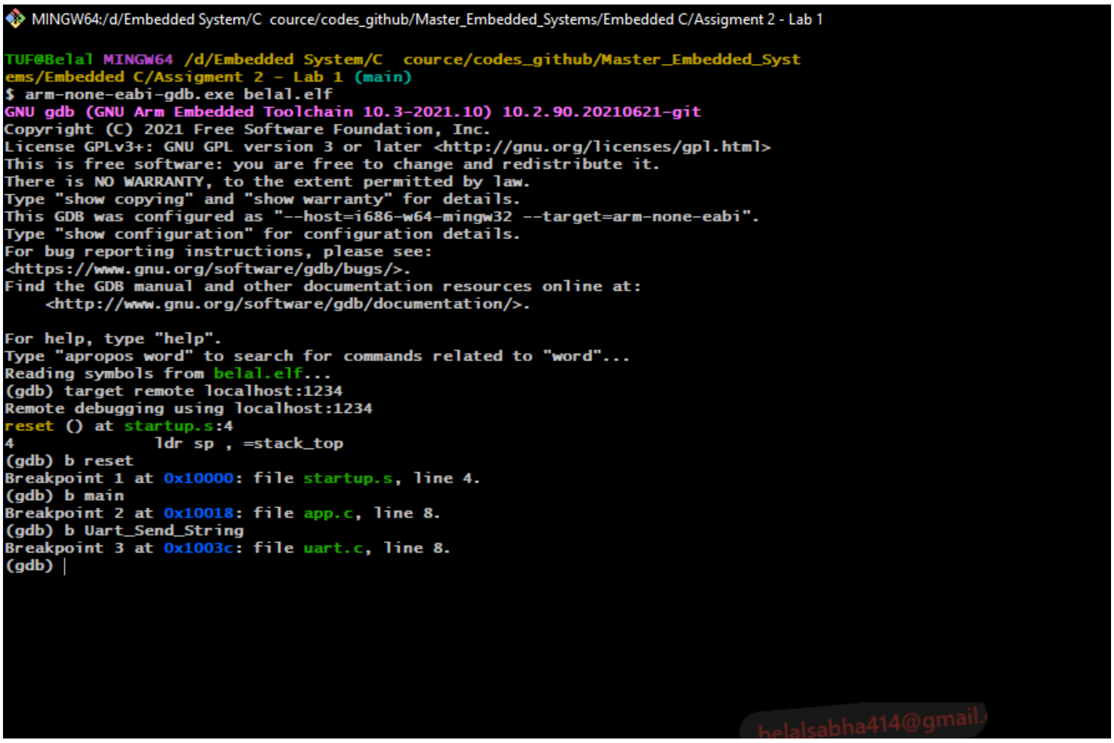
```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-ld.exe -T linker.ld app.o uart.o startup.o -o belal.elf

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ |
```

5- Create .bin file :

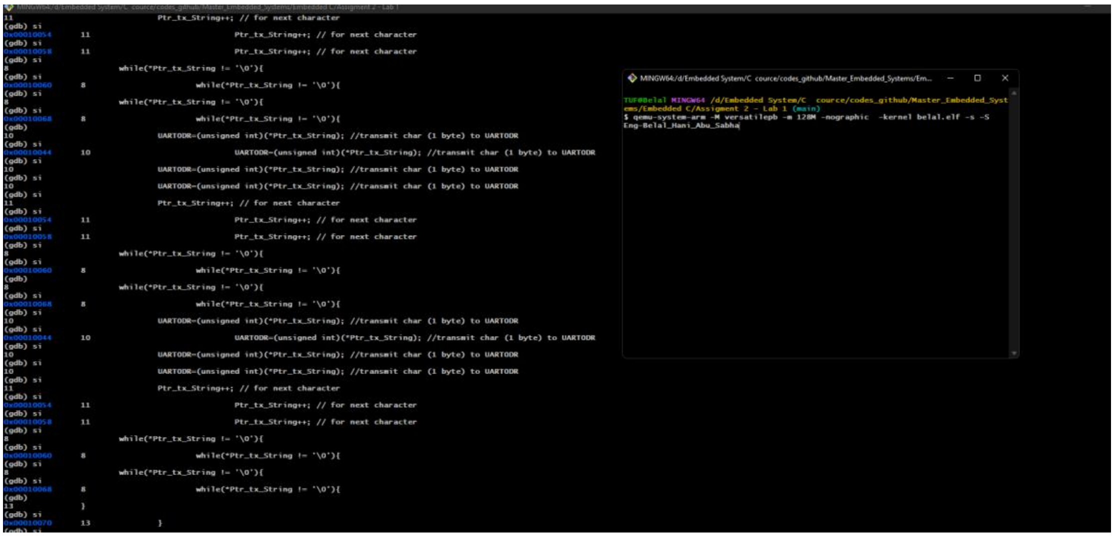

MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - L...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ arm-none-eabi-ld.exe -T linker.ld app.o uart.o startup.o -o belal.elf

TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Systems/Embedded C/As
sigment 2 - Lab 1 (main)
$ |
```

6-Use GDB to debug the program and find if there is any problem or not then display output.


MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - Lab 1

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ arm-none-eabi-gdb.exe belal.elf
GNU gdb (GNU Arm Embedded Toolchain 10.3-2021.10) 10.2.90.20210621-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from belal.elf...
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:4
4               ldr sp , =stack_top
(gdb) b reset
Breakpoint 1 at 0x10000: file startup.s, line 4.
(gdb) b main
Breakpoint 2 at 0x10018: file app.c, line 8.
(gdb) b Uart_Send_String
Breakpoint 3 at 0x1003c: file uart.c, line 8.
(gdb) |
```



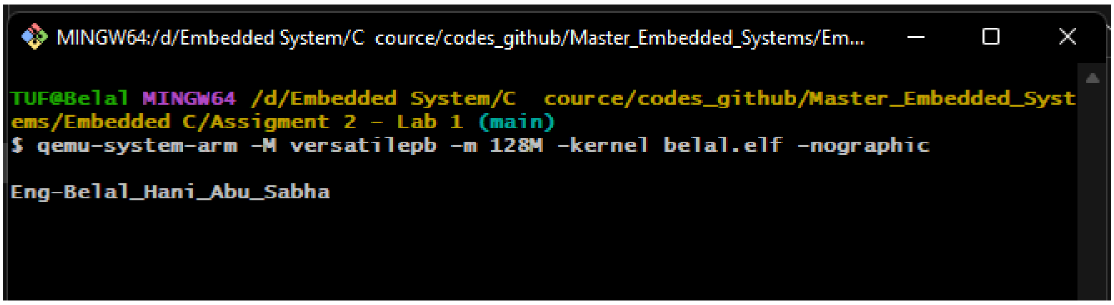
MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Embedded C/Assigment 2 - Lab 1

```
11                              Ptr_tx_String++; // for next character
(gdb) si
0x00010054      11                              Ptr_tx_String++; // for next character
(gdb) si
0x00010058      11                              Ptr_tx_String++; // for next character
(gdb) si
8                       while(*Ptr_tx_String != '\0'){
(gdb) si
0x00010060      8                       while(*Ptr_tx_String != '\0'){
(gdb) si
8                       while(*Ptr_tx_String != '\0'){
(gdb) si
0x00010068      8                       while(*Ptr_tx_String != '\0'){
(gdb)
10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
0x00010044      10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
11                              Ptr_tx_String++; // for next character
(gdb) si
0x00010054      11                              Ptr_tx_String++; // for next character
(gdb) si
0x00010058      11                              Ptr_tx_String++; // for next character
(gdb) si
8                       while(*Ptr_tx_String != '\0'){
(gdb) si
0x00010060      8                       while(*Ptr_tx_String != '\0'){
(gdb)
8                       while(*Ptr_tx_String != '\0'){
(gdb) si
0x00010068      8                       while(*Ptr_tx_String != '\0'){
(gdb) si
10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
0x00010044      10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
10                              UARTODR=(unsigned int)(*Ptr_tx_String); //transmit char (1 byte) to UARTODR
(gdb) si
11                              Ptr_tx_String++; // for next character
(gdb) si
0x00010054      11                              Ptr_tx_String++; // for next character
(gdb) si
0x00010058      11                              Ptr_tx_String++; // for next character
(gdb) si
8                       while(*Ptr_tx_String != '\0'){
(gdb) si
0x00010060      8                       while(*Ptr_tx_String != '\0'){
(gdb) si
8                       while(*Ptr_tx_String != '\0'){
(gdb) si
0x00010068      8                       while(*Ptr_tx_String != '\0'){
(gdb)
13              }
(gdb) si
0x00010070      13              }
(gdb) si
```

MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Em...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ qemu-system-arm -M versatilepb -m 128M -nographic  -kernel belal.elf -s -S
Eng-Belal_Hani_Abu_Sabha
```

7- Code run successfully on Arm926Ej-S chip and output should be :



MINGW64:/d/Embedded System/C cource/codes_github/Master_Embedded_Systems/Em...

```
TUF@Belal MINGW64 /d/Embedded System/C  cource/codes_github/Master_Embedded_Syst
ems/Embedded C/Assigment 2 - Lab 1 (main)
$ qemu-system-arm -M versatilepb -m 128M -kernel belal.elf -nographic

Eng-Belal_Hani_Abu_Sabha
```