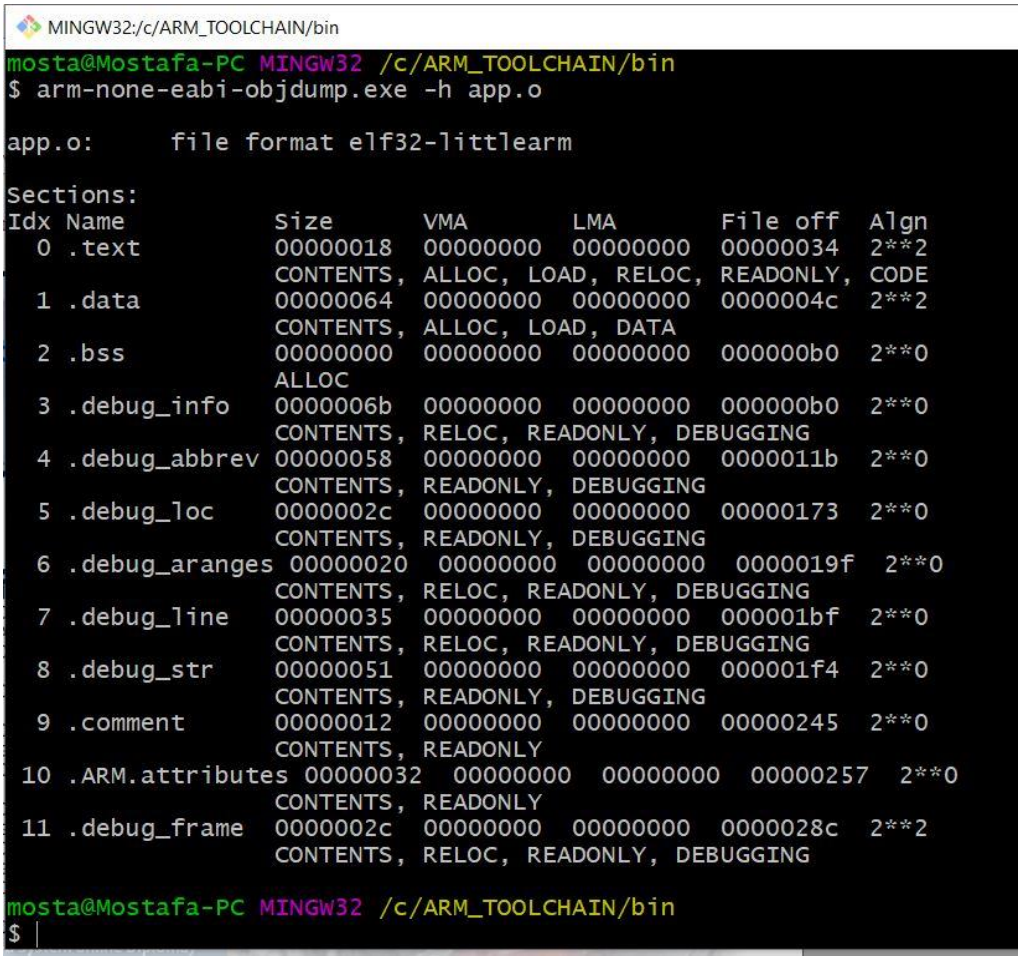
LAB 1 :

- First creating files and writing code for the application (app.c, uart.c and uart.h) then compiling the files and generating (app.o and uart.o) files:

- Then take a look at code sections with debug for app.o:



And for uart.o:

MINGW32:/c/ARM_TOOLCHAIN/bin

```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ 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 .debug_info   0000005c  00000000  00000000  00000084  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 00000051  00000000  00000000  000000e0  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000002c  00000000  00000000  00000131  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  0000015d  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   0000003d  00000000  00000000  0000017d  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    00000050  00000000  00000000  000001ba  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  0000020a  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000032  00000000  00000000  0000021c  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000028  00000000  00000000  00000250  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$
```

- Then generating assembly files for app.o and uart.o:

MINGW32:/c/ARM_TOOLCHAIN/bin

```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-objdump.exe -D app.o >> app.asm

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-objdump.exe -D uart.o >> uart.asm

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$
```

- Creating startup.s,write startup code and then pass it to assembler and getting startup.o file:

MINGW32:/c/ARM_TOOLCHAIN/bin

```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o
startup.s: Assembler messages:
startup.s:4: Warning: partial line at end of file ignored

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$
```

- Taking a look at code sections for starup.o:

MINGW32:/c/ARM_TOOLCHAIN/bin

```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ 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

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$
```

- Removing debug section and looking at app.o and uart.o code sections afterwards:

MINGW32:/c/ARM_TOOLCHAIN/bin

```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ 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

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$
```

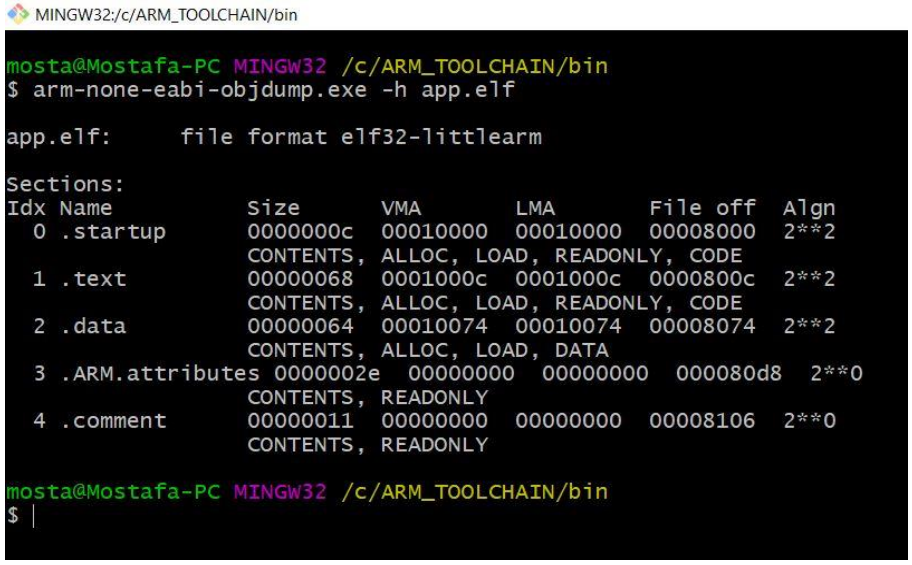
```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-objdump.exe -h app.o

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

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$
```

- Then it’s time to write the linker script and link all files together to get app.elf and generate the map_file.map:

MINGW32:/c/ARM_TOOLCHAIN/bin

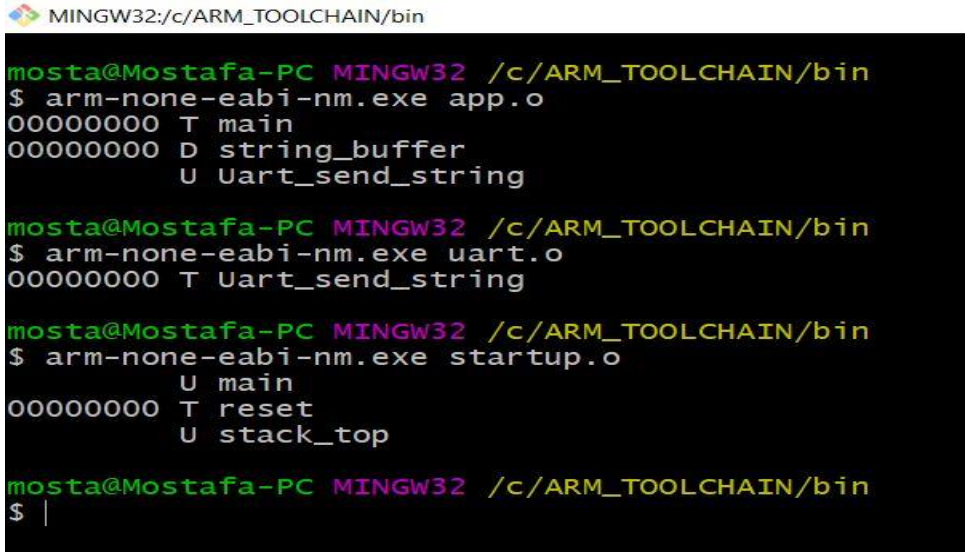
```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-ld.exe -T linker_script.ld startup.o app.o uart.o -o app.elf -Map=Map_file.map

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$
```

- Take a look at the sections in app.elf:

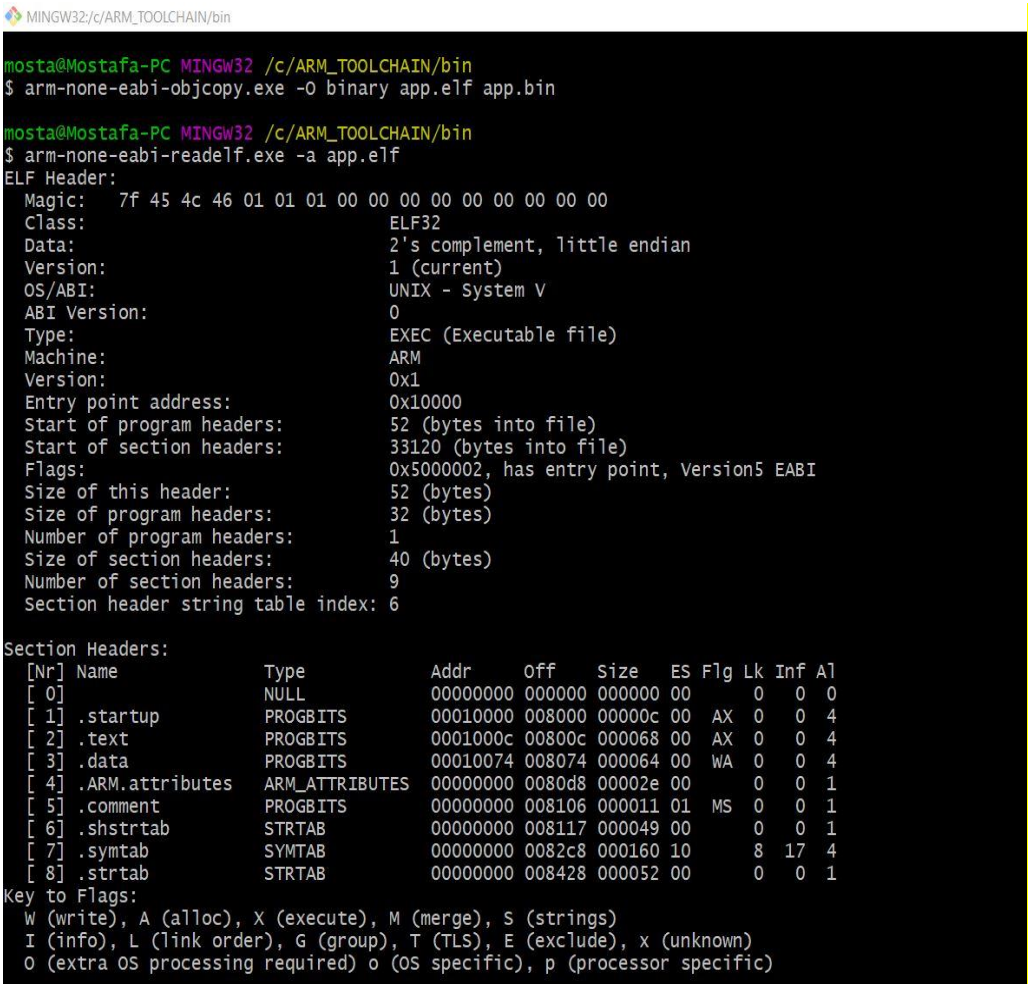


- Now take a look at the symbols in the app.elf:

- Also take a look at symbols in app.o, uart.o and startup.o:



- Now we read the elf file created earlier to make sure that the entry point address is correct and that the startup code is at that address:



- Now that everything looks fine the code is to be tested on the Quick Emulator (qemu):