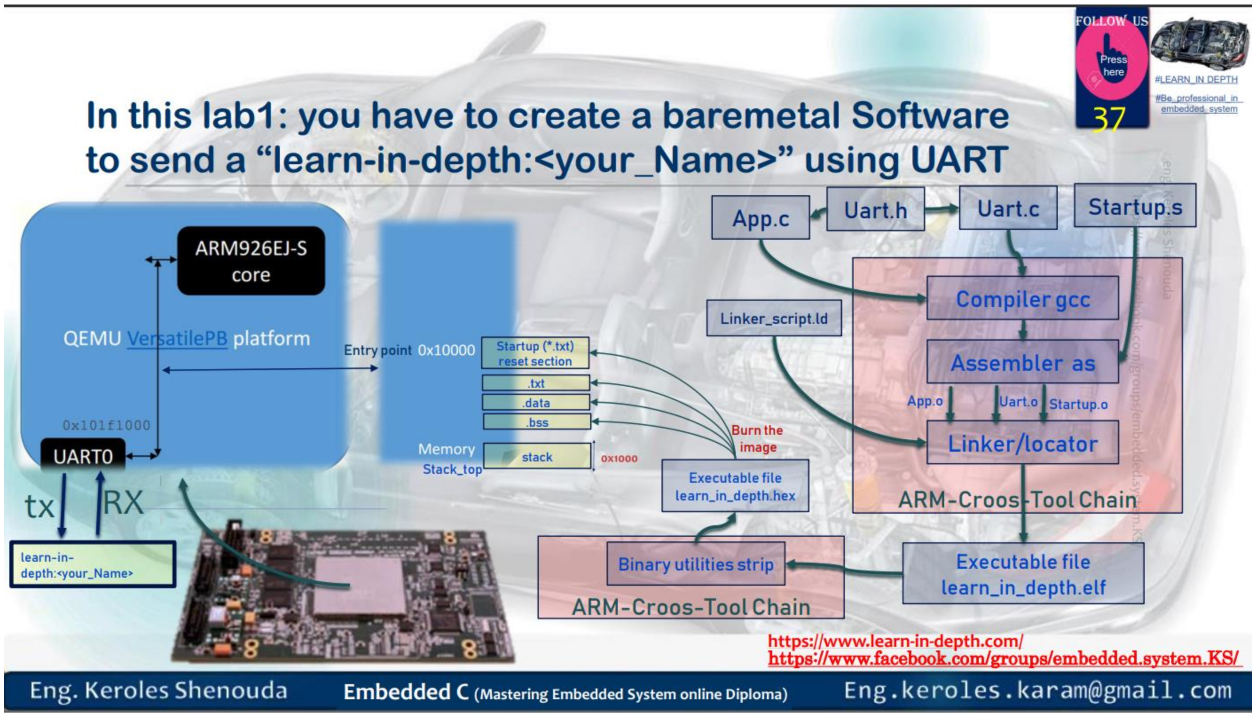
Saeed Mabrouk Saeed El-Shaikh

A simple application of executing a bare metal application on ARM VersatilePB

⇨ The program is to send “learn-in-depth: Saeed Fares” to the terminal through the UART0 peripheral

⇨ I will achieve that by going on some steps

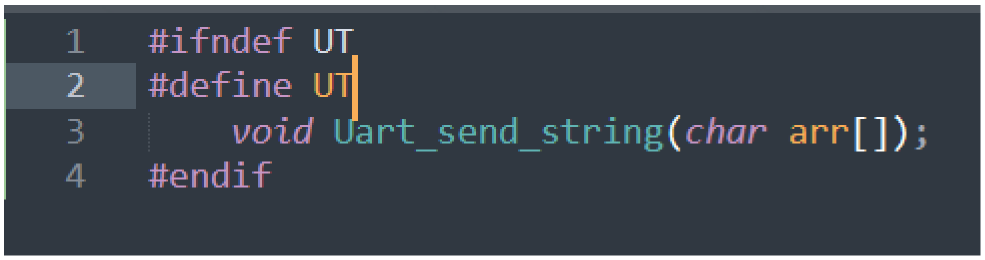
1) Write C codes which achieve this task and compile it to get object files
2) Write startup.s file and feed it to the compiler and get object file
3) Write the linker_script file to give it the linker
4) Use linker to link these object files and generate the binary file
5) burn the binary file in the VersatilePB to run the code



In this lab1: you have to create a baremetal Software to send a “learn-in-depth:<your_Name>” using UART
FOLLOW US
Press here
37
#LEARN_IN DEPTH
#Be_professional_in_embedded_system
ARM926EJ-S core
QEMU VersatilePB platform
0x101f1000
UART0
tx
RX
learn-in-depth:<your_Name>
Entry point 0x10000
Startup (*.txt) reset section
.txt
.data
.bss
Memory
Stack_top
stack
0x1000
Burn the image
Executable file learn_in_depth.hex
App.c
Uart.h
Uart.c
Startup.s
Linker_script.ld
Compiler gcc
Assembler as
App.o
Uart.o
Startup.o
Linker/locator
ARM-Croos-Tool Chain
Binary utilities strip
ARM-Croos-Tool Chain
Executable file learn_in_depth.elf
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/
Eng. Keroles Shenouda
Embedded C (Mastering Embedded System online Diploma)
Eng.keroles.karam@gmail.com

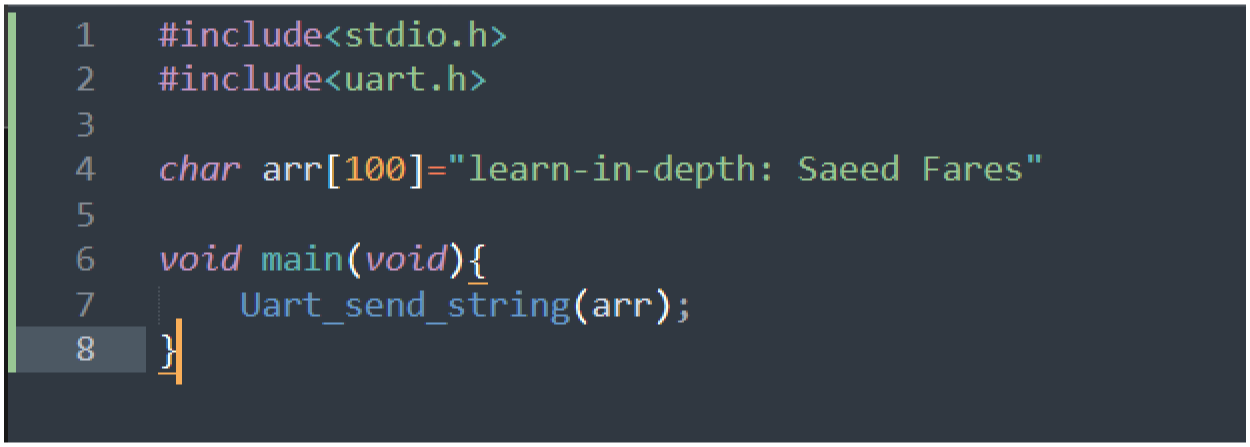
1) *Write C codes which achieve this task*

Uart.h



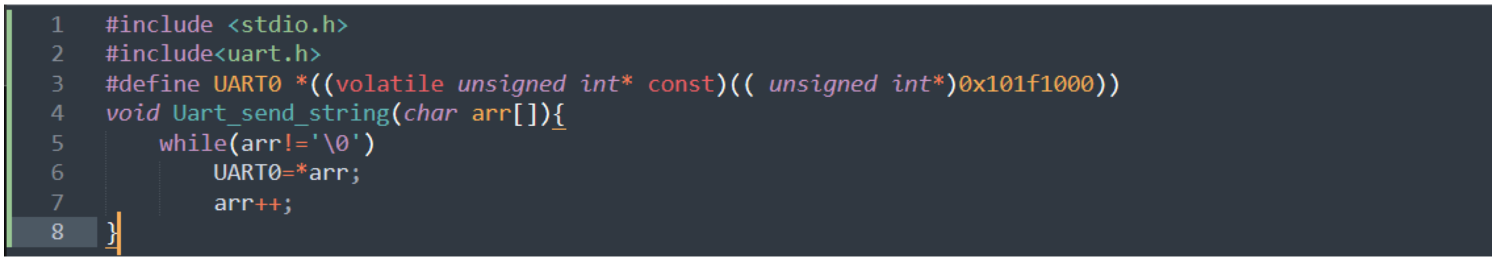
```
#ifndef UT
#define UT
    void Uart_send_string(char arr[]);
#endif
```

Uart.c



```c
#include<stdio.h>
#include<uart.h>

char arr[100]="learn-in-depth: Saeed Fares"

void main(void){
    Uart_send_string(arr);
}
```

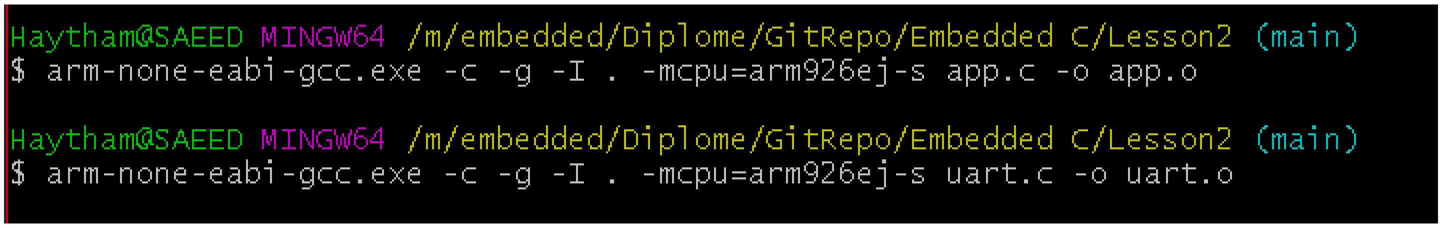
App.c



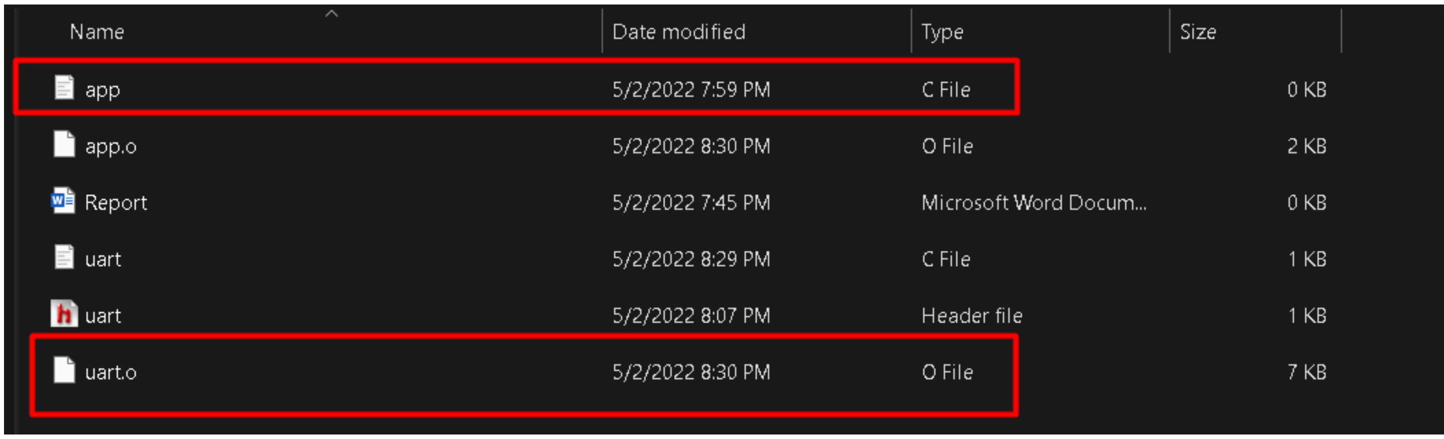
```c
#include <stdio.h>
#include<uart.h>
#define UART0 *((volatile unsigned int* const)(( unsigned int*)0x101f1000))
void Uart_send_string(char arr[]){
    while(arr!='\0')
        UART0=*arr;
        arr++;
}
```

$arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s app.c -o app.o

- -c ➔ compile and assemple only not link
- -g ➔ generate debug information
- -I ➔include header
- . ➔ the header in the same directory I'm in
- -mcpu= arm926ej-s ➔ machine
- -o ➔output to file



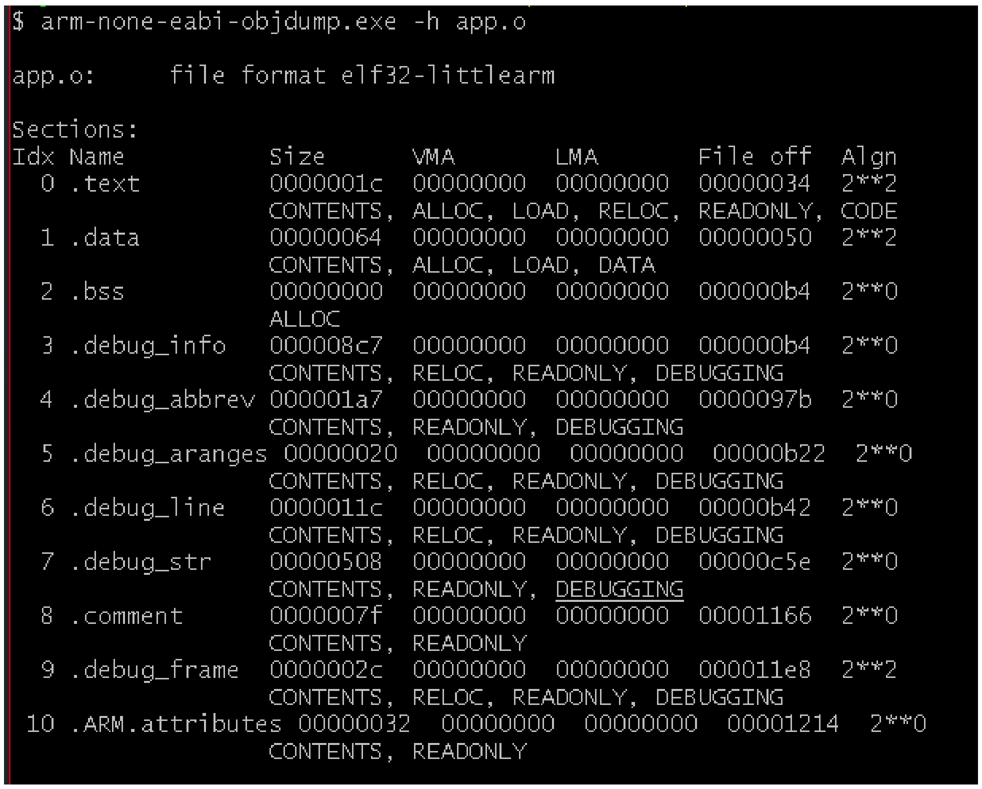
```
Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s app.c -o app.o

Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s uart.c -o uart.o
```



Name	Date modified	Type	Size
app	5/2/2022 7:59 PM	C File	0 KB
app.o	5/2/2022 8:30 PM	O File	2 KB
Report	5/2/2022 7:45 PM	Microsoft Word Docum...	0 KB
uart	5/2/2022 8:29 PM	C File	1 KB
uart	5/2/2022 8:07 PM	Header file	1 KB
uart.o	5/2/2022 8:30 PM	O File	7 KB

Navigate object files

1- App.o



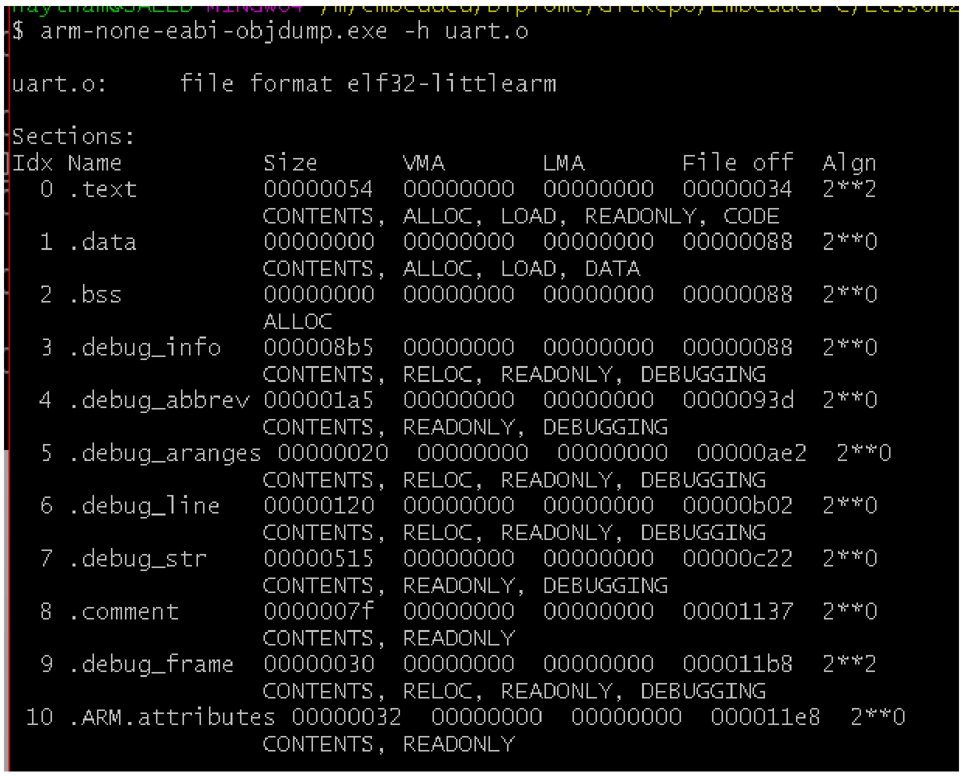
```
$ arm-none-eabi-objdump.exe -h app.o

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         00000064  00000000  00000000  00000050  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b4  2**0
                  ALLOC
  3 .debug_info   000008c7  00000000  00000000  000000b4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001a7  00000000  00000000  0000097b  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_aranges 00000020  00000000  00000000  00000b22  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  6 .debug_line   0000011c  00000000  00000000  00000b42  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_str    00000508  00000000  00000000  00000c5e  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .comment      0000007f  00000000  00000000  00001166  2**0
                  CONTENTS, READONLY
  9 .debug_frame  0000002c  00000000  00000000  000011e8  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
 10 .ARM.attributes 00000032  00000000  00000000  00001214  2**0
                  CONTENTS, READONLY
```

data section 64 hexa===100 byte which is arr[100]

bss section 0 because we not have uninitialized variables

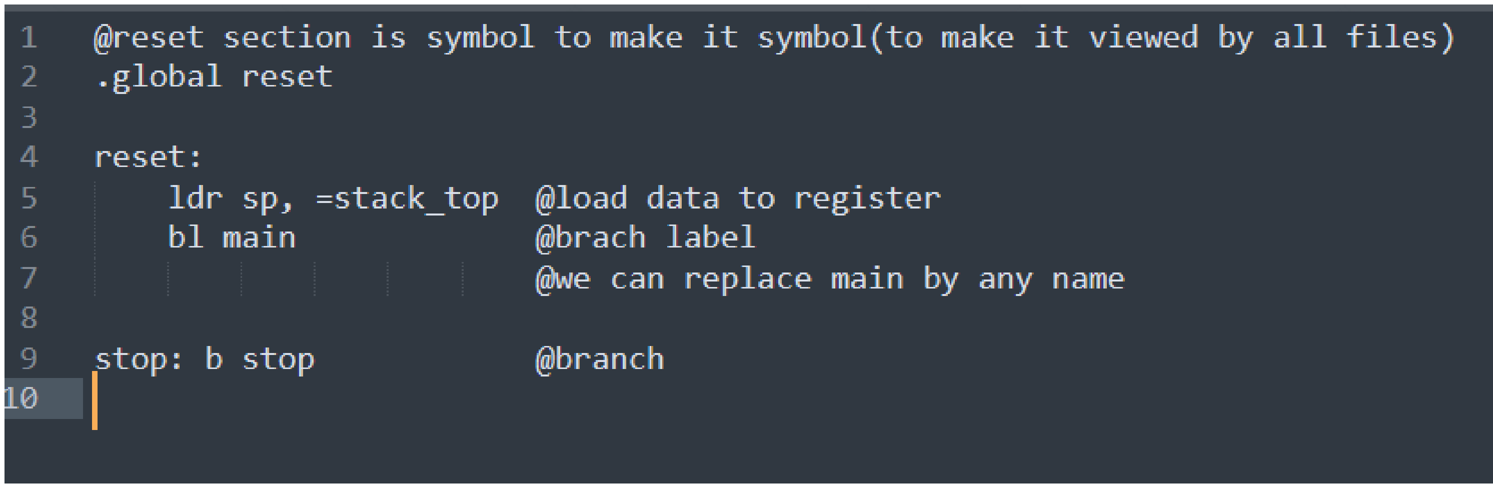
all addresses is 0 because onbect files are allocatable files and physical addresses is set during linker stage

2- uart.o



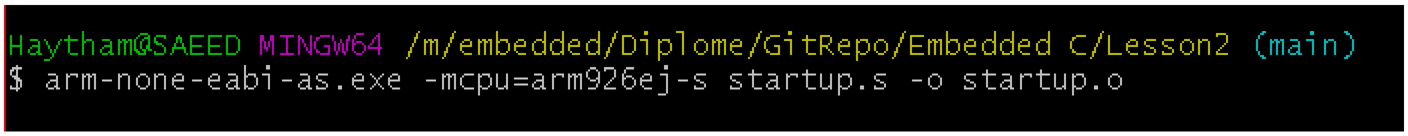
```
$ arm-none-eabi-objdump.exe -h uart.o

uart.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000054  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000088  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000088  2**0
                  ALLOC
  3 .debug_info   000008b5  00000000  00000000  00000088  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001a5  00000000  00000000  0000093d  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_aranges 00000020  00000000  00000000  00000ae2  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  6 .debug_line   00000120  00000000  00000000  00000b02  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_str    00000515  00000000  00000000  00000c22  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .comment      0000007f  00000000  00000000  00001137  2**0
                  CONTENTS, READONLY
  9 .debug_frame  00000030  00000000  00000000  000011b8  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
 10 .ARM.attributes 00000032  00000000  00000000  000011e8  2**0
                  CONTENTS, READONLY
```

2) *Write startup.s file*



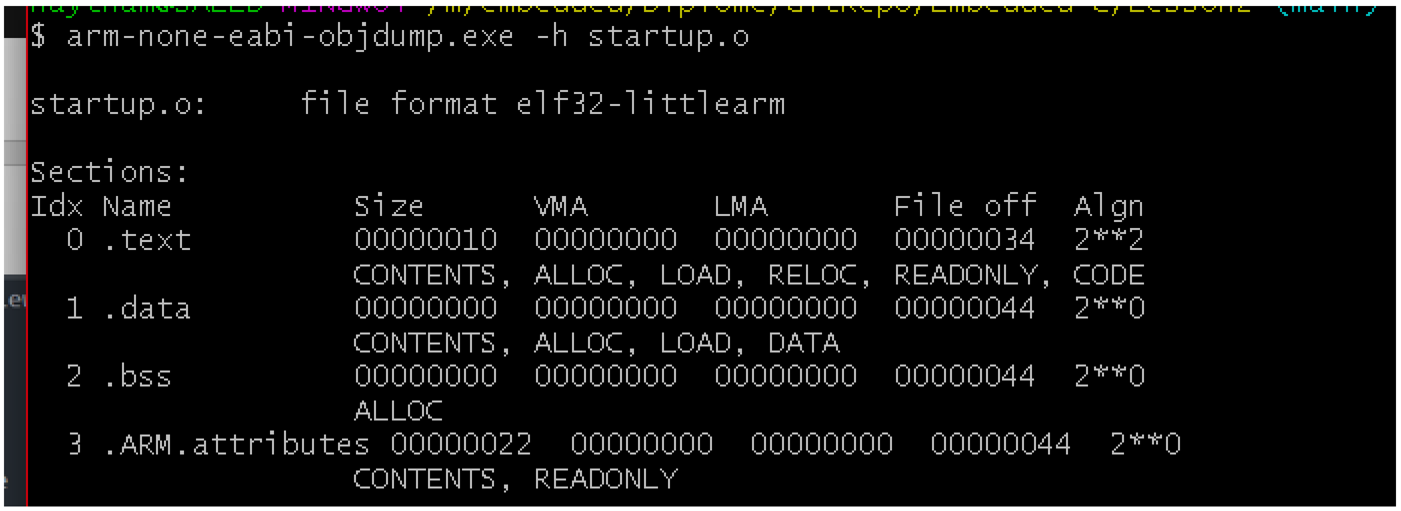
```
@reset section is symbol to make it symbol(to make it viewed by all files)
.global reset

reset:
    ldr sp, =stack_top  @load data to register
    bl main             @brach label
                        @we can replace main by any name

stop: b stop            @branch

```

Compile startup file



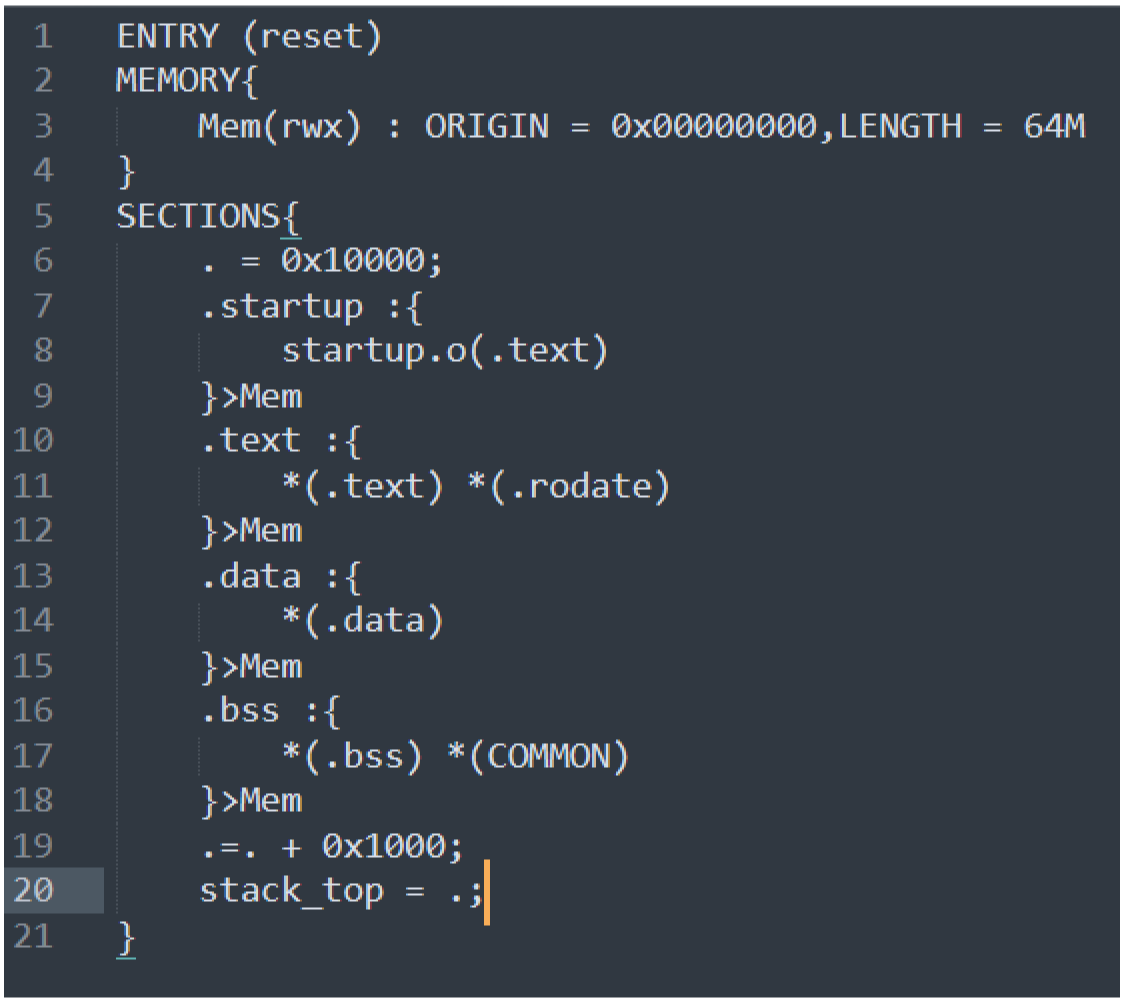
```
Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o
```

Navigiation



```
$ 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
```

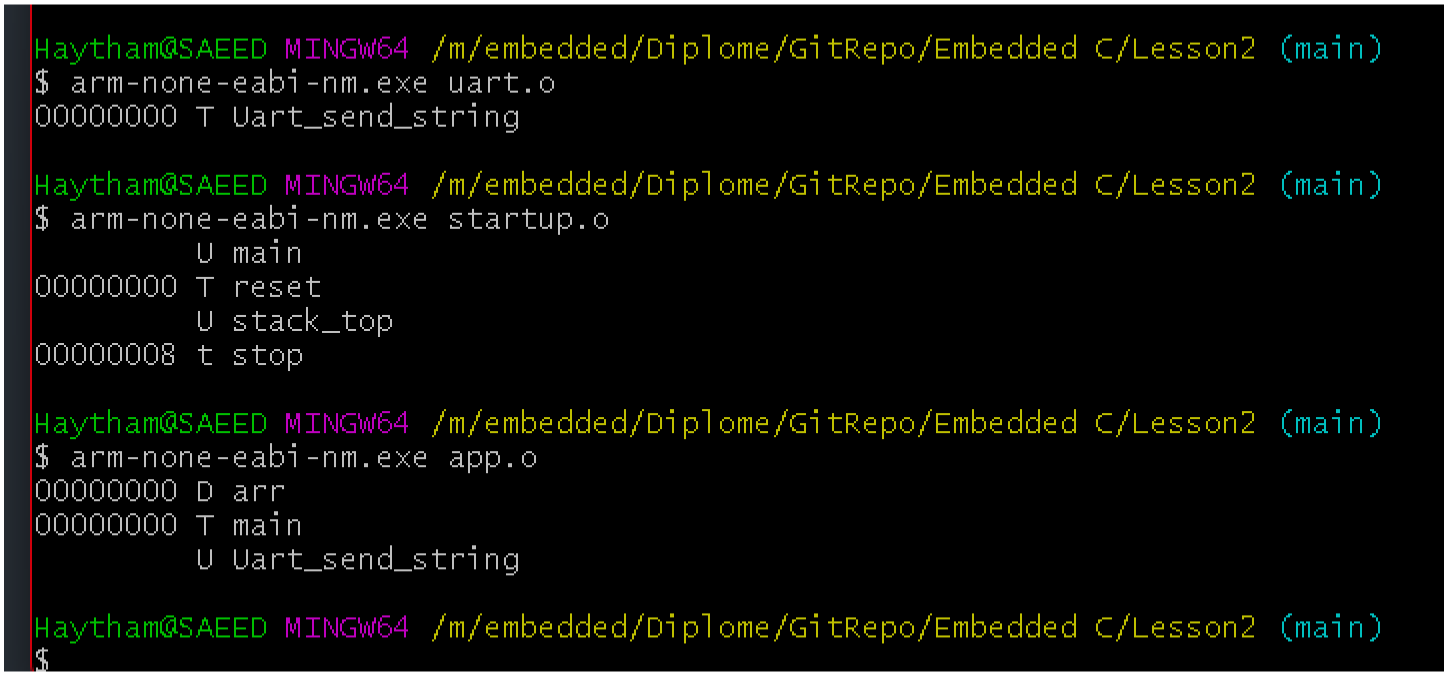
3) *Writing linker script*



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

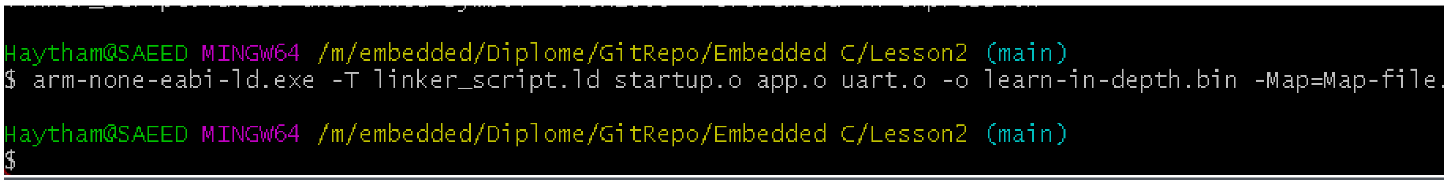
Checking symbols in object files

The addresses is zero because object files are reallocate

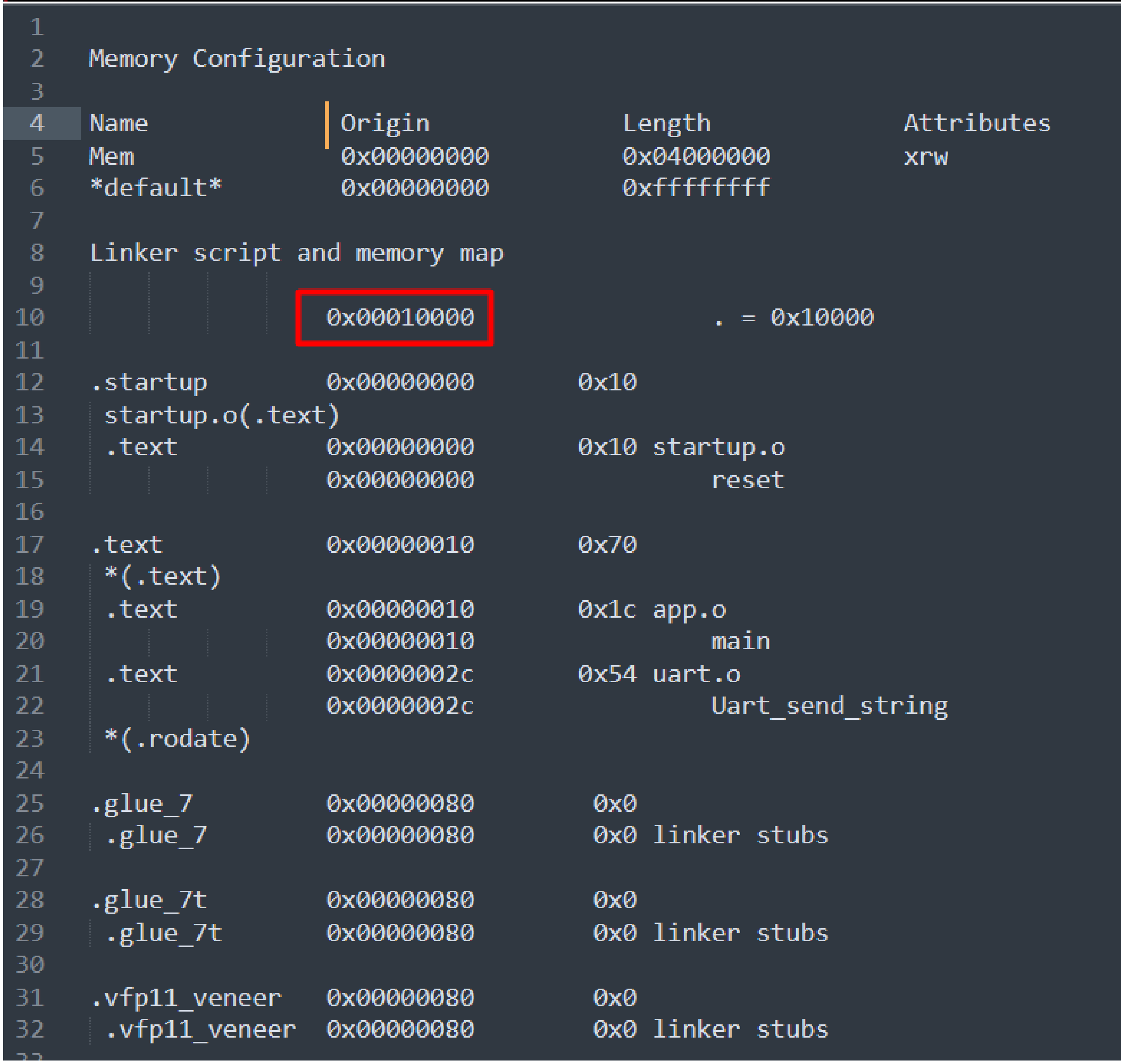


```
Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-nm.exe uart.o
00000000 T Uart_send_string

Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-nm.exe startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop

Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-nm.exe app.o
00000000 D arr
00000000 T main
         U Uart_send_string

Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$
```

4) Use linker to link these object files and generate the binary file



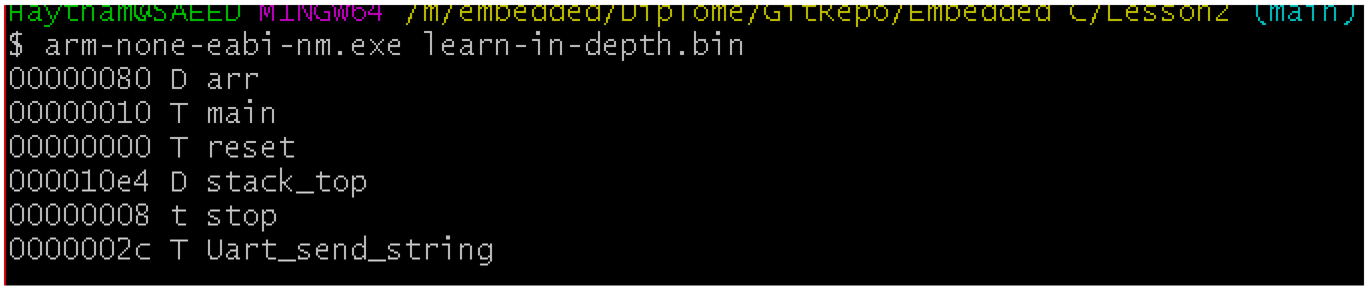
```
Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-ld.exe -T linker_script.ld startup.o app.o uart.o -o learn-in-depth.bin -Map=Map-file.

Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$
```



```

Memory Configuration

Name             Origin             Length             Attributes
Mem              0x00000000         0x04000000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map

                0x00010000                . = 0x10000

.startup        0x00000000       0x10
 startup.o(.text)
 .text          0x00000000       0x10 startup.o
                0x00000000                reset

.text           0x00000010       0x70
 *(.text)
 .text          0x00000010       0x1c app.o
                0x00000010                main
 .text          0x0000002c       0x54 uart.o
                0x0000002c                Uart_send_string
 *(.rodate)

.glue_7         0x00000080        0x0
 .glue_7        0x00000080        0x0 linker stubs

.glue_7t        0x00000080        0x0
 .glue_7t       0x00000080        0x0 linker stubs

.vfp11_veneer   0x00000080        0x0
 .vfp11_veneer  0x00000080        0x0 linker stubs
```

In map file it's shown the startup section(reset) in the entry point 0x00010000

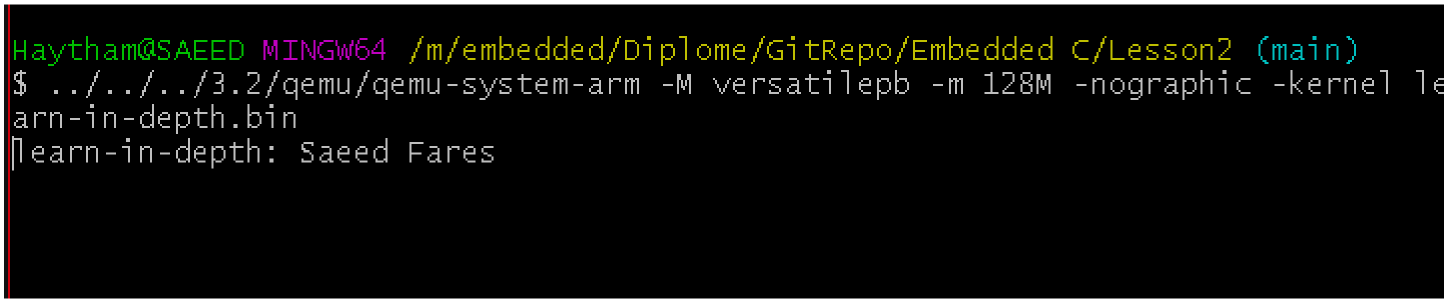
Symbols in the .exe file



```
Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ arm-none-eabi-nm.exe learn-in-depth.bin
00000080 D arr
00000010 T main
00000000 T reset
000010e4 D stack_top
00000008 t stop
0000002c T Uart_send_string
```

Real addresses are shown here

5) burn the binary file in the VersatilePB to run the code



```
Haytham@SAEED MINGW64 /m/embedded/Diplome/GitRepo/Embedded C/Lesson2 (main)
$ ../../../3.2/qemu/qemu-system-arm -M versatilepb -m 128M -nographic -kernel le
arn-in-depth.bin
learn-in-depth: Saeed Fares
```