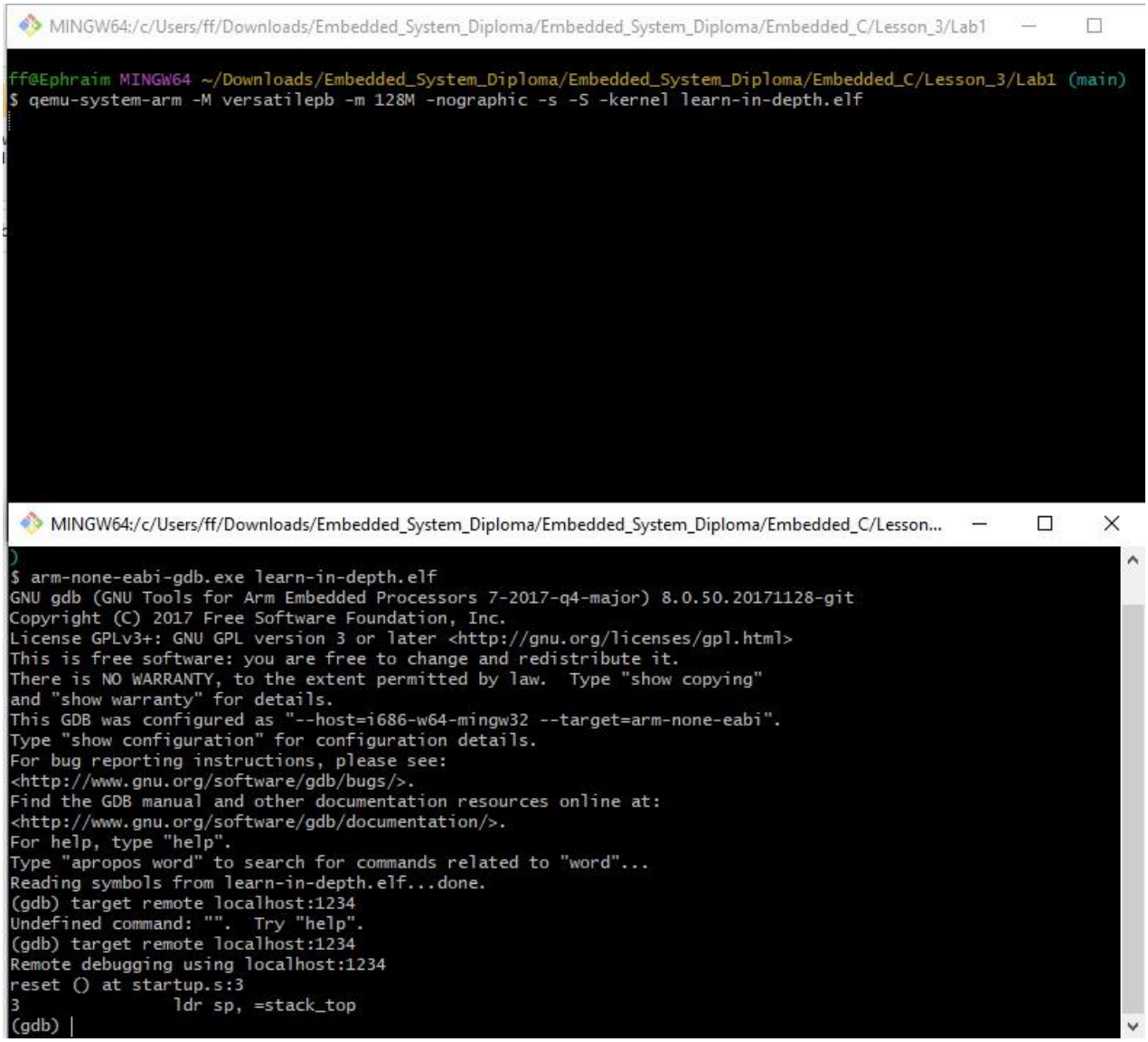
LESSON 3 LABS

Guirguis Hedia

Debugging Steps with Terminal :

- Start Debugging



- Commands Used to Debug

```
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab1 (main)
$ arm-none-eabi-gdb.exe learn-in-depth.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 8.0.50.20171128-
git
Copyright (C) 2017 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:
<http://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 learn-in-depth.elf...done.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:3
3               ldr sp, =stack_top
(gdb) l
1       .globl reset
2       reset:
3               ldr sp, =stack_top
4               bl main
5       stop:   b       stop
(gdb) b main
Breakpoint 1 at 0x10018: file app.c, line 7.
(gdb) b *0x10010
Breakpoint 2 at 0x10010: file app.c, line 6.
(gdb) s
reset () at startup.s:4
4               bl main
(gdb) s

Breakpoint 2, main () at app.c:6
6       {
(gdb) s

Breakpoint 1, main () at app.c:7
7               Uart_Send_String (string_buffer);
(gdb) print string_buffer[0]
$1 = 108 'l'
(gdb) watch string_buffer
Hardware watchpoint 3: string_buffer
(gdb)
```

To make Breakpoint at The beginning of C code of Main functioin

To make breakpoint at The Beginning of main Function "Not The Beginning of C Code"

Command to Step C Code line

Command to Step C Code line

Command to Step C Code line

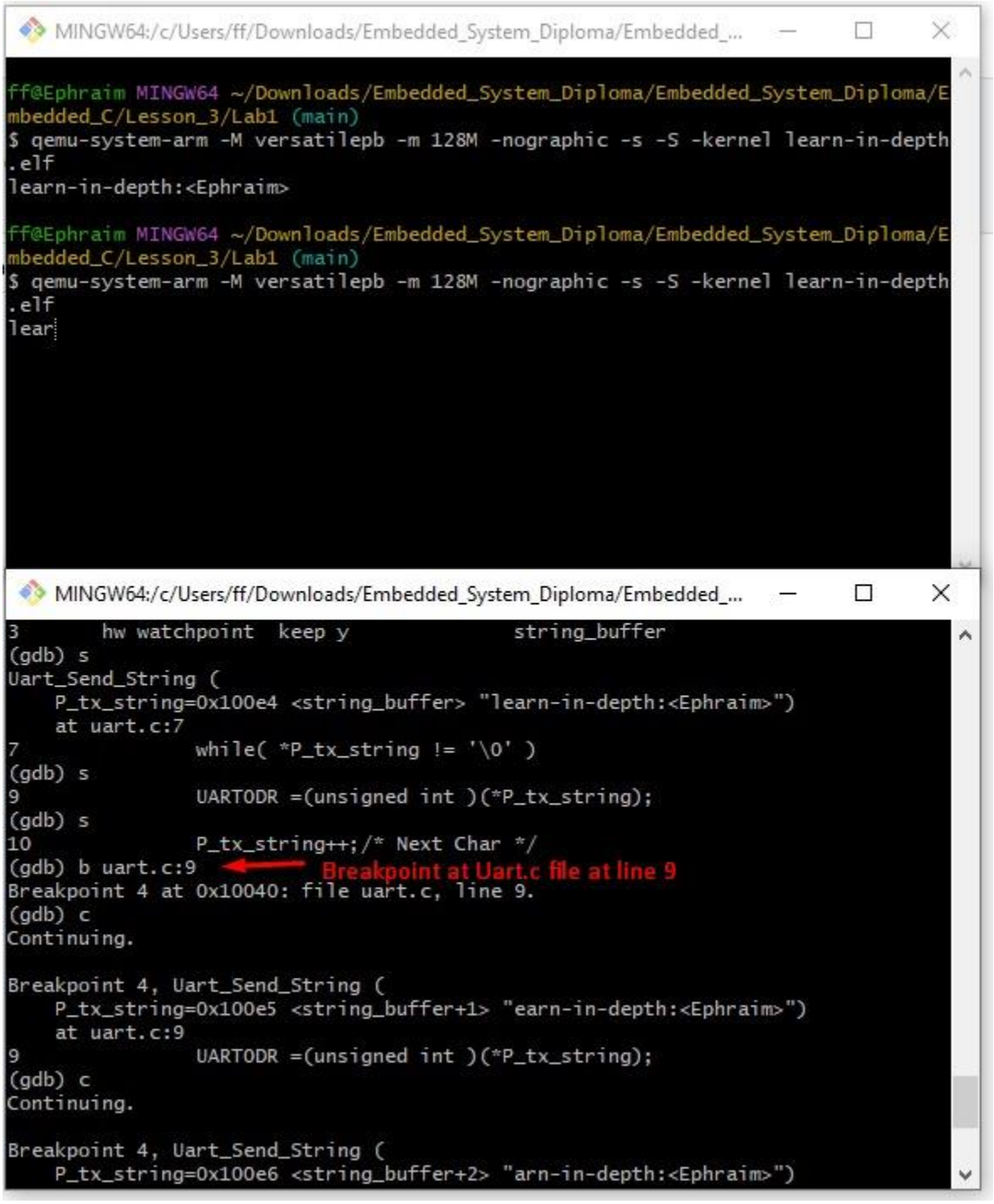
To print String_buffer[0] Value

```
(gdb) watch string_buffer
Hardware watchpoint 3: string_buffer
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x00010018 in main at app.c:7
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00010010 in main at app.c:6
        breakpoint already hit 1 time
3       hw watchpoint  keep y              string_buffer
```

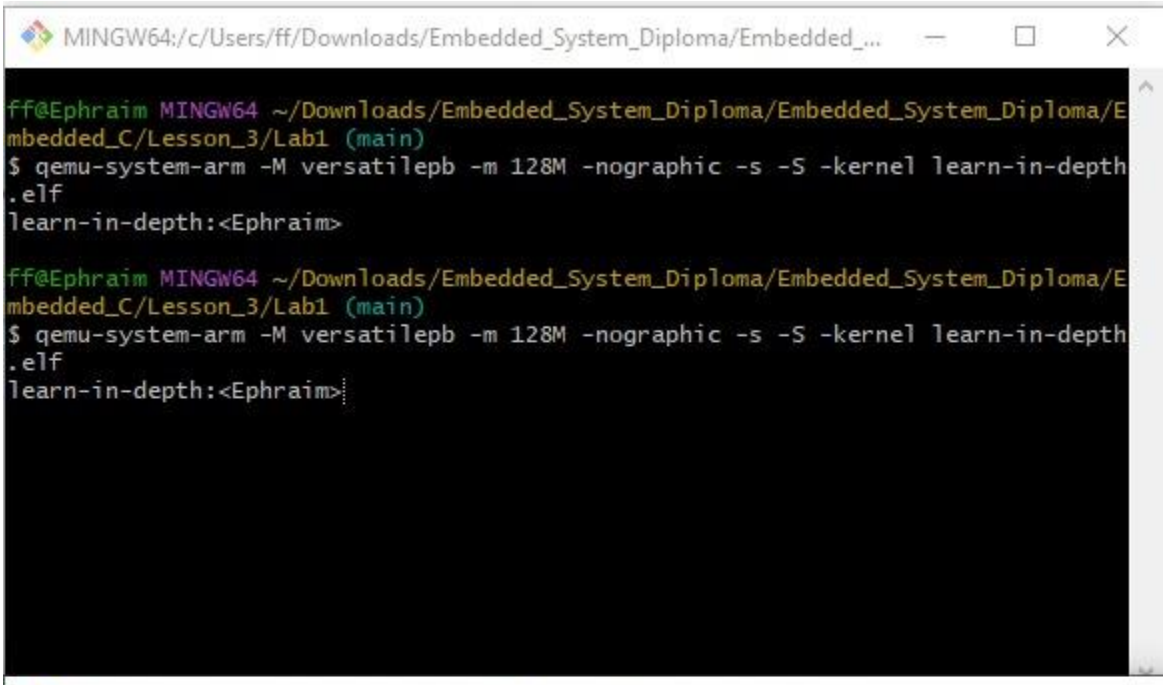
To Watch The Symbol string_buffer if Change

To Watch The Beakpoints information

- The Output During Debug



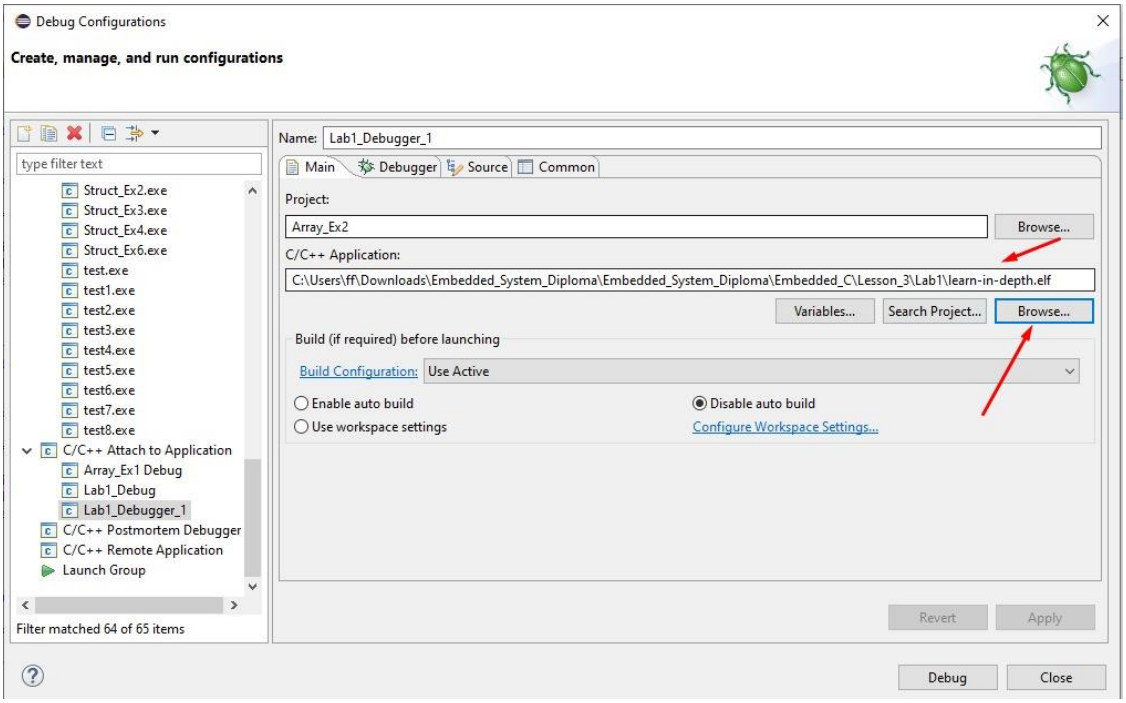
- Debug is Finished And The Message is Printed



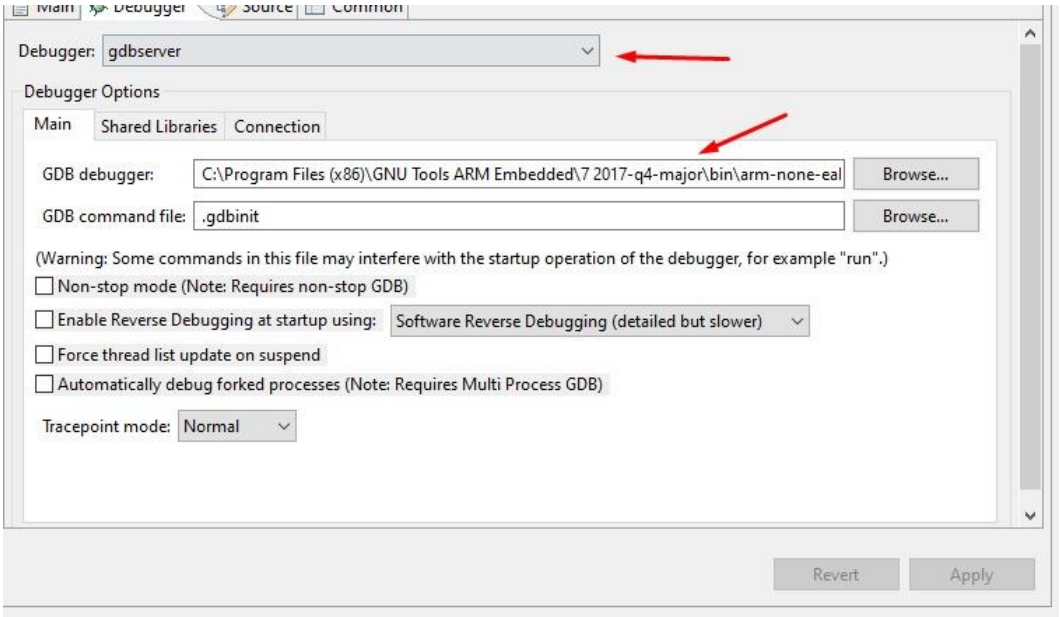
Debugging Steps with Eclipse :

- Debug Configuration to Start Debugging in Eclipse

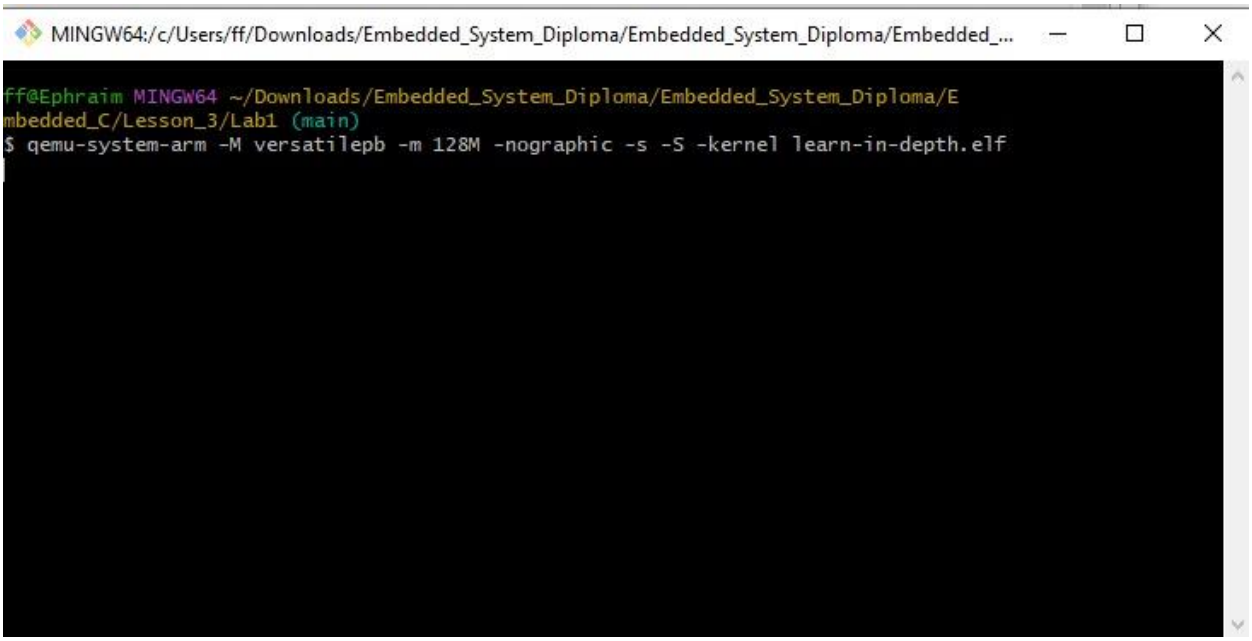
1. Choose Any Project file and Select The Real elf file you want to Debug



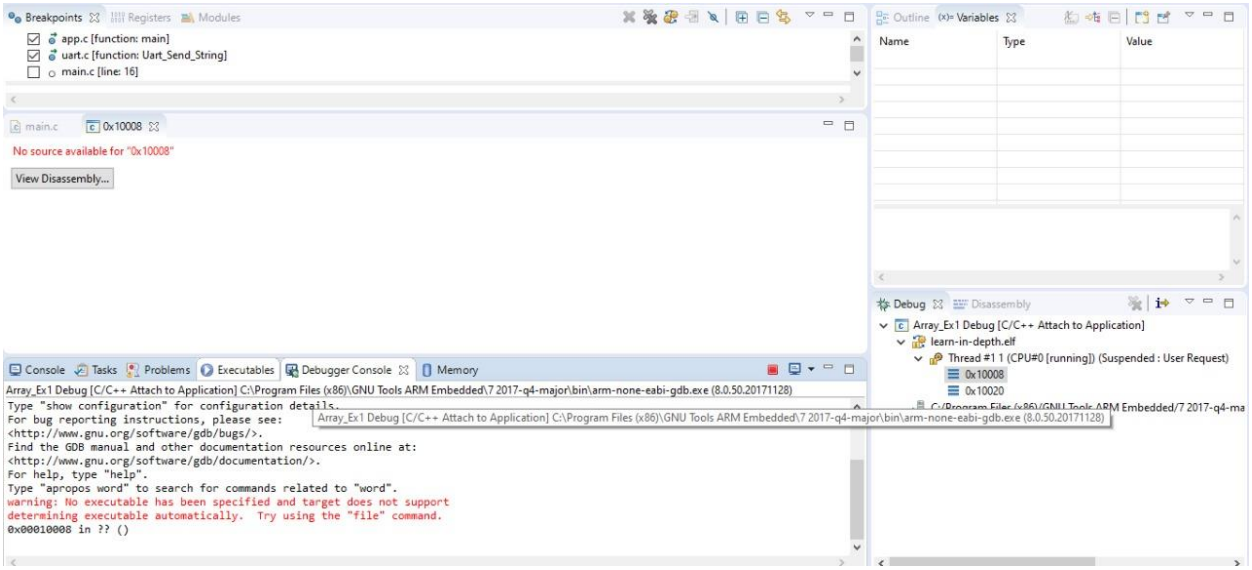
2. Choose gdbserver option and the arm toolchain



3. Write a Command To Open The Board Virsually in Qemu



4. Start Debug In Eclipse But you Will Find This Error



To Solve This Error

you should put The elf File in The Project Folder you choose

Then write file Command To Start Debugging

5. To See The Project file Direction Path

   Write pwd command to see The path of The project you choose

   Then Put the elf File in The folder

```
undefined command: "rwd". Try "help".
pwd
Working directory C:\Users\ff\workspace\C_Programming\Array_Ex3.
```

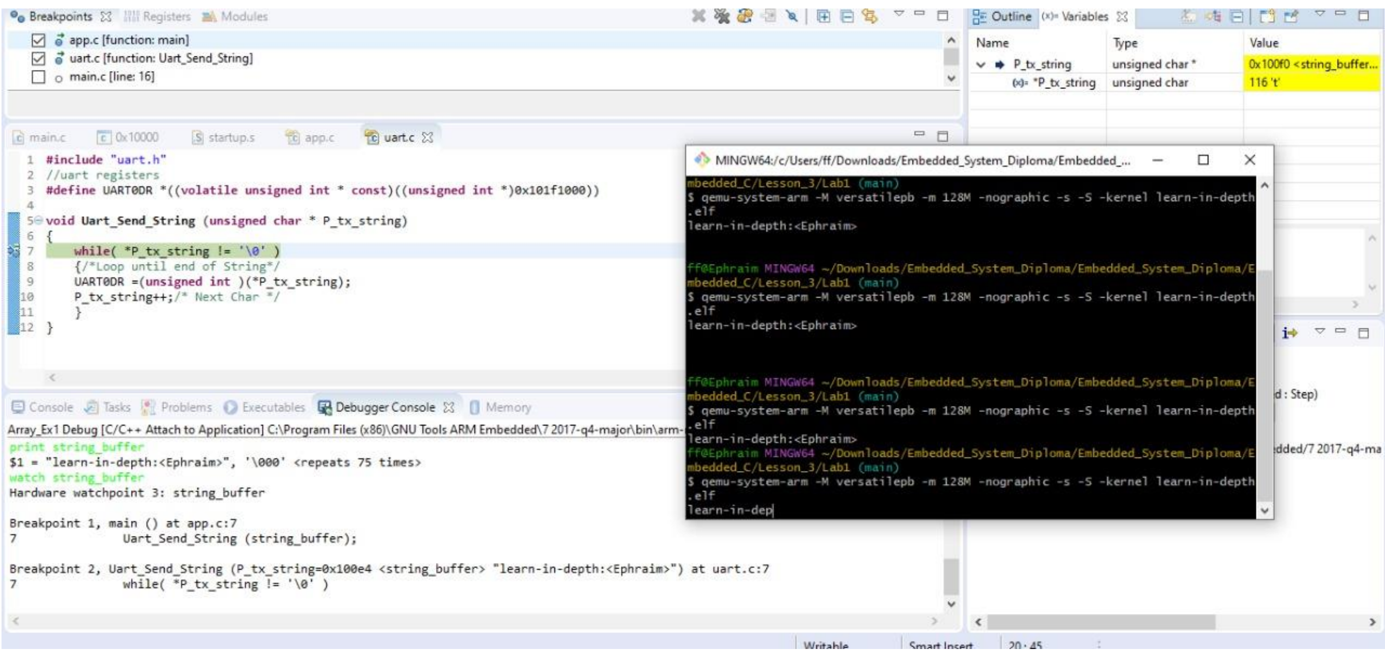
Then Copy the elf file to This path

6. Use File Command To Read Symbols and Start debug

```
Working directory C:\Users\ff\workspace\C_Programming\Array_Ex3.
file learn-in-depth.elf
A program is being debugged already.
Are you sure you want to change the file? (y or n) [answered Y; input not from terminal]
Reading symbols from learn-in-depth.elf...done.
```

Use file Command to Read Symbol

7. The Result During Debug



Lesson 3 Lab 2 with Startup.s file :

- Main.c file

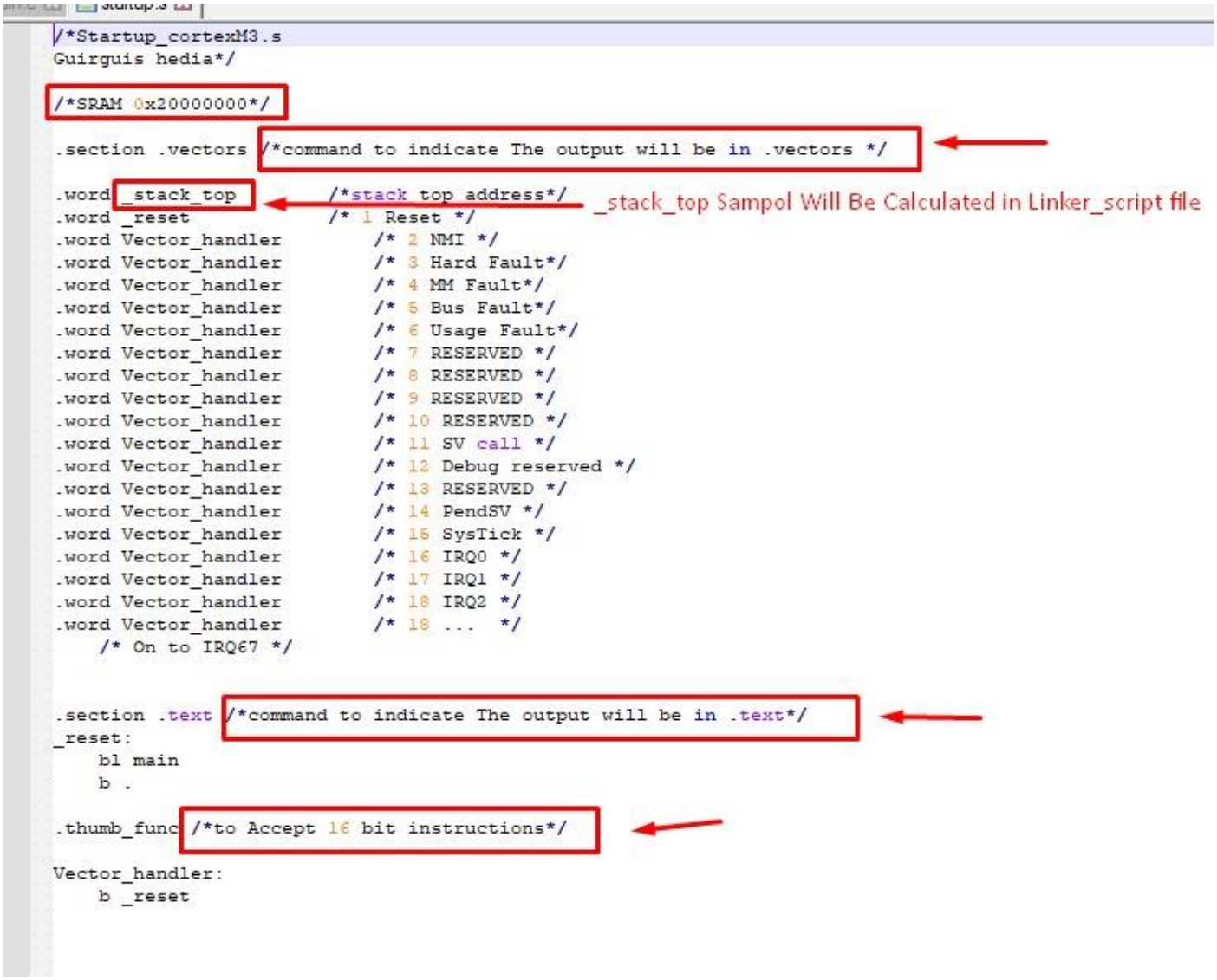
```c
#define RCC_BASE      0x40021000
#define GPIOA_BASE    0x40010800
#define RCC_APB2ENR   *(volatile uint32_t *) (RCC_BASE+0x18)
#define GPIOA_CRH     *(volatile uint32_t *) (GPIOA_BASE+0x04)
#define GPIOA_ODR     *(volatile uint32_t *) (GPIOA_BASE+0x0C)
//Bit Fields
#define RCC_IOPAEN   (1<<2)
#define GPIOA13      (1UL<<13)


typedef union {
        vint32_t    all_fields;
        struct {
        vint32_t    reserved:13 ;
        vint32_t    pin13:1 ;
        }pin;
}R_ODR_t;

volatile R_ODR_t* R_ODR=(volatile R_ODR_t*)(GPIOA_BASE+0x0C);
unsigned char g_variables[3]={1,2,3};
unsigned char const const_variables[3]={1,2,3};
unsigned char volatile bss_variables[3];

int main(void)
{
    RCC_APB2ENR |=RCC_IOPAEN;
    GPIOA_CRH &=0xFF0FFFFF  ;
    GPIOA_CRH |=0x00200000 ;
    while(1)
    {
//      GPIO_ODR |=(1<<13); //Set Bit 13
        R_ODR->pin.pin13=1;
        for (int i=0 ;i<5000;i++);

//      GPIO_ODR &=~(1<<13); //Clear Bit 13
        R_ODR->pin.pin13=0;
        for (int i=0 ;i<5000;i++);

    }

    return 0;
}
```

To Increase .data Section Size

To Create .rodata Section

To Create Size For .bss Section

- Startup.s file



- Linker_Script file

```
/*Linker Script CortexM3
Guirguis Hedia
*/
MEMORY
{
flash(RX)  : ORIGIN =0x08000000, LENGTH =128K
sram(RWX)  : ORIGIN =0x20000000, LENGTH =20K

}

SECTIONS
{
    .text : {
            *(.vectors*)
            *(.text*)
            *(.rodata)
            _E_text = .;
    }>flash


    .data : {
    _S_DATA = . ;
    *(.data)
    . = ALIGN(4) ;
    _E_DATA = . ;

    }>flash

    .bss : {
    _S_bss = . ;
    *(.bss*)
    _E_bss = . ;
    . = ALIGN(4);
    . = . +0x1000;
    _stack_top = .;

    }> sram
}
```

flash memory will begin at address 0x08000000 with Size 128k

SRAM memory will begin ad address 0x2000000 with Size 20 k

The first Address here in 0x08000000 store The address which be used by Sp

The processor use this address to jump to Stack_Top in SRAM

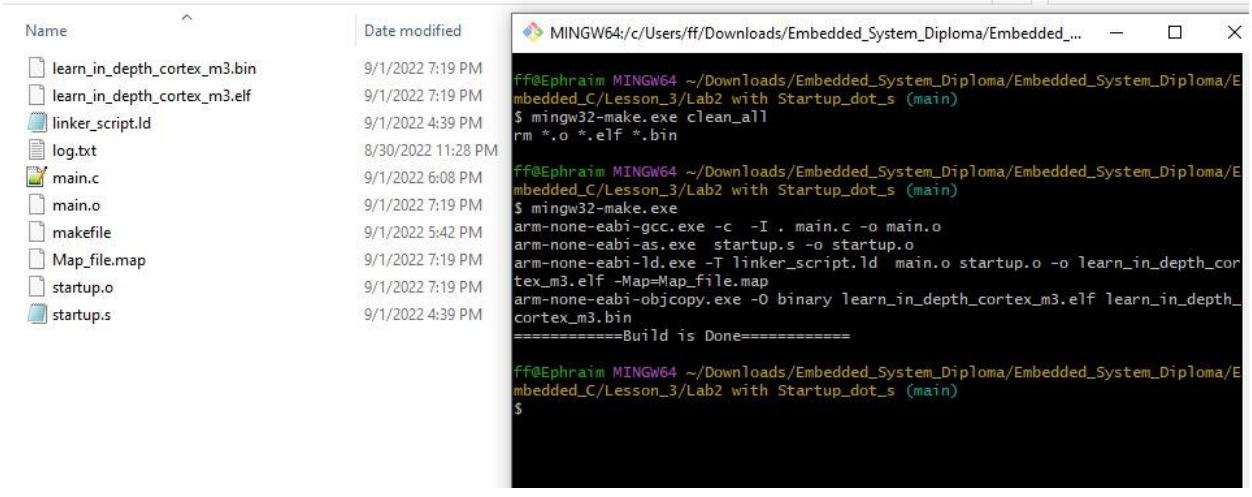
This Commad To Make The Memory Alignment
To Increase The Preformance

Stack_top Will be Located after .bss Section with Size =4K Byte
This Will Be Shown in Map_file.map

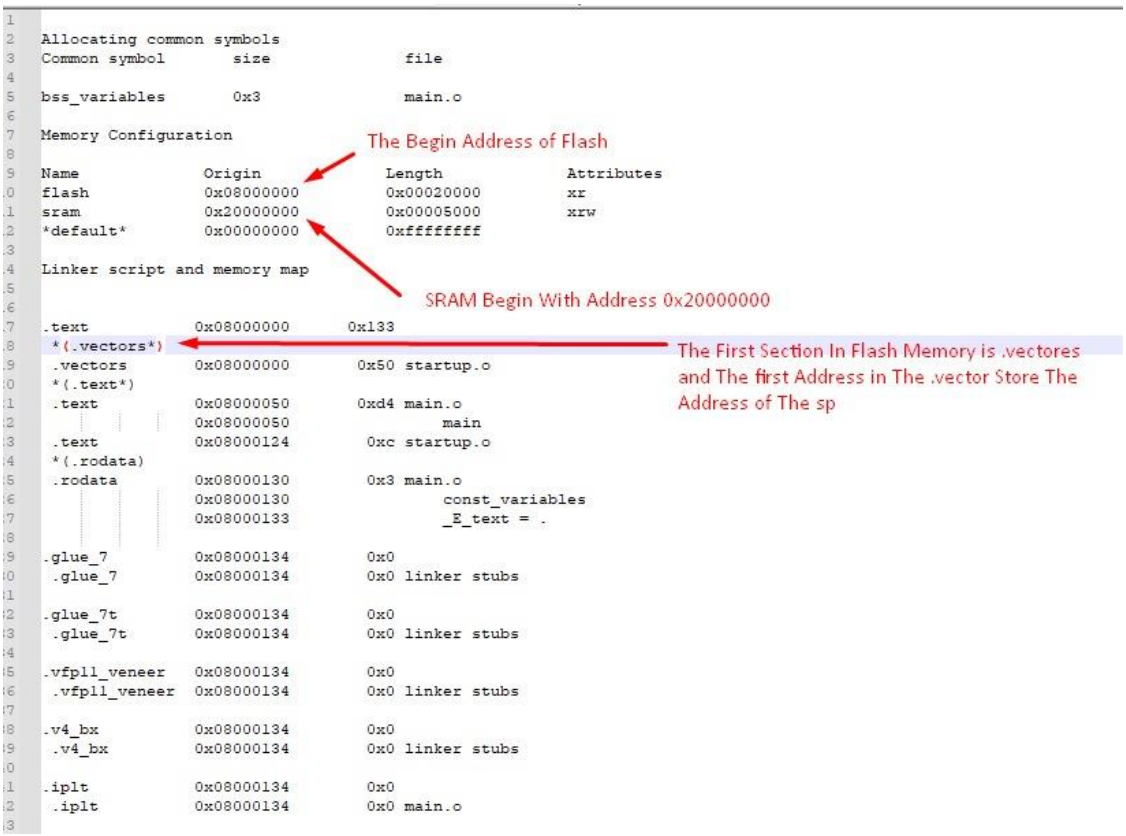
- Make file

```
#@Copyright : Guirguis Hedia

CC=arm-none-eabi-
CFLAG=-gdwarf-2 -mcpu=cortex-m3   Processor Name
INCS=-I .
LIBS=
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
As= $(wildcard *.s)
AsOBJ= $(As:.s=.o)

Project_Name=learn_in_depth_cortex_m3

all: $(Project_Name).bin
    @echo "============Build is Done============"

startup.o: startup.s
    $(CC)as.exe $(CFLAGS) $< -o $@

%.o: %.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@


$(Project_Name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsOBJ) -o $@ -Map=Map_file.map   To Generate Map_file.map

$(Project_Name).bin: $(Project_Name).elf

    $(CC)objcopy.exe -O binary $< $@

clean_all:
        rm *.o *.elf *.bin

clean:
        rm *.elf *.bin
```

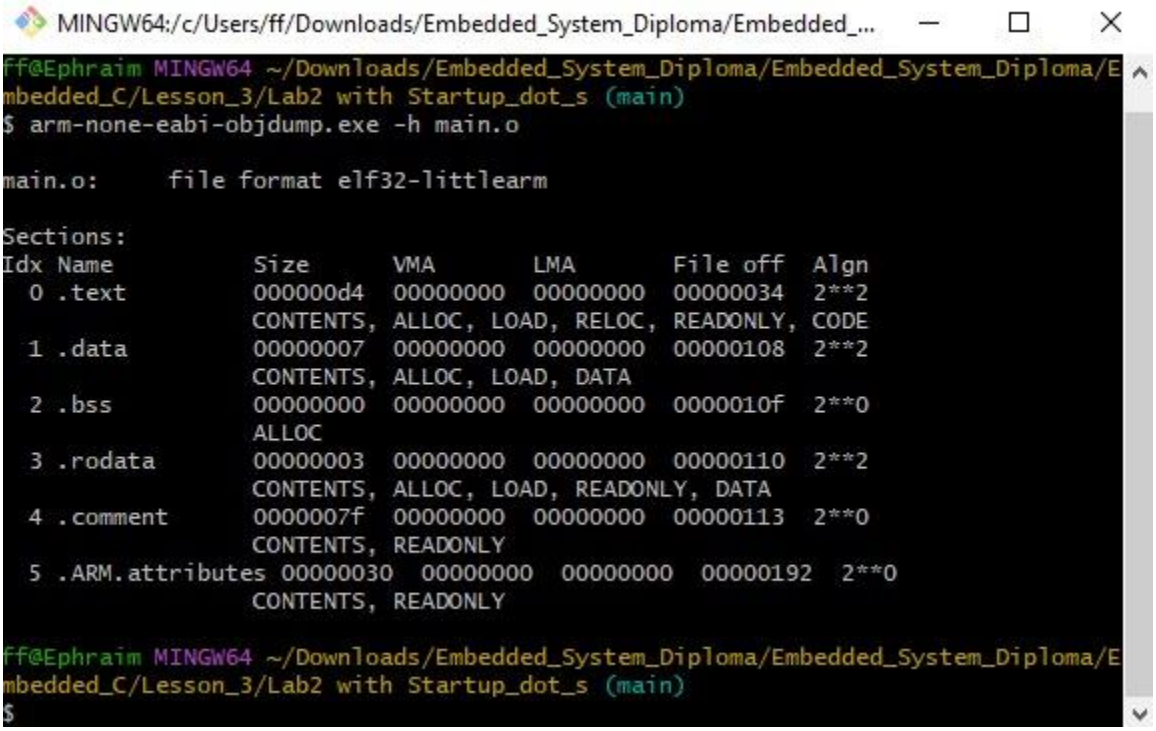
- Building is Done



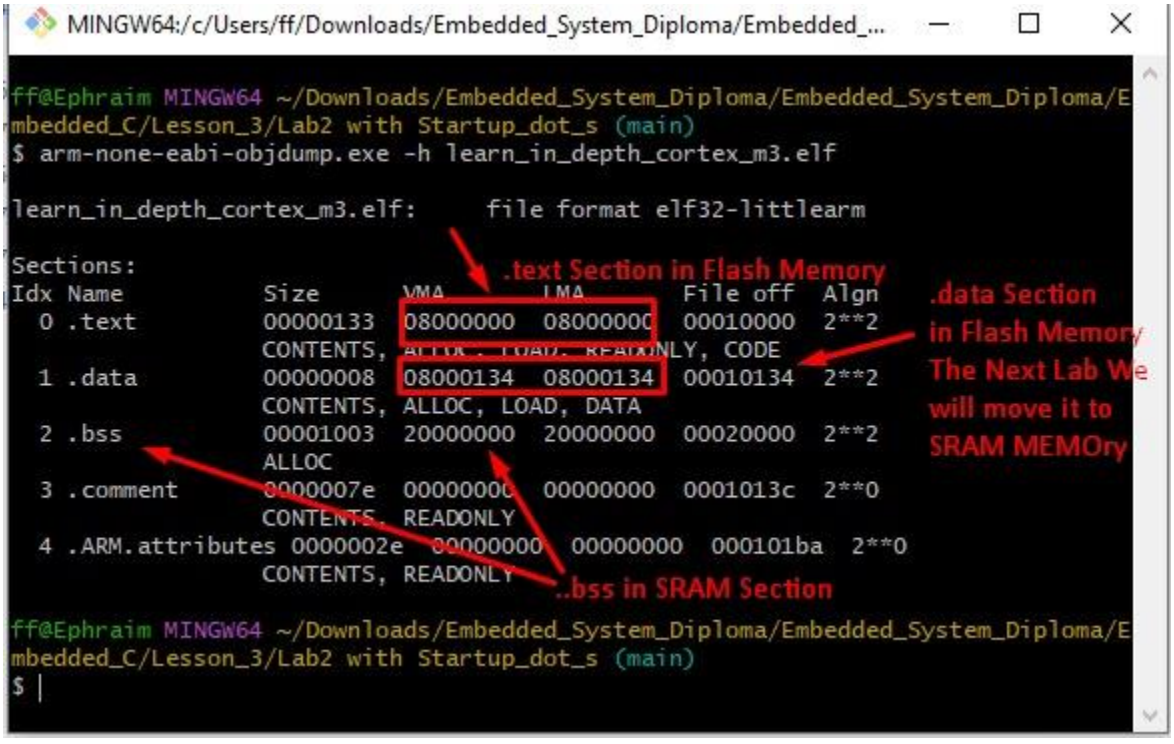
- Map File Details



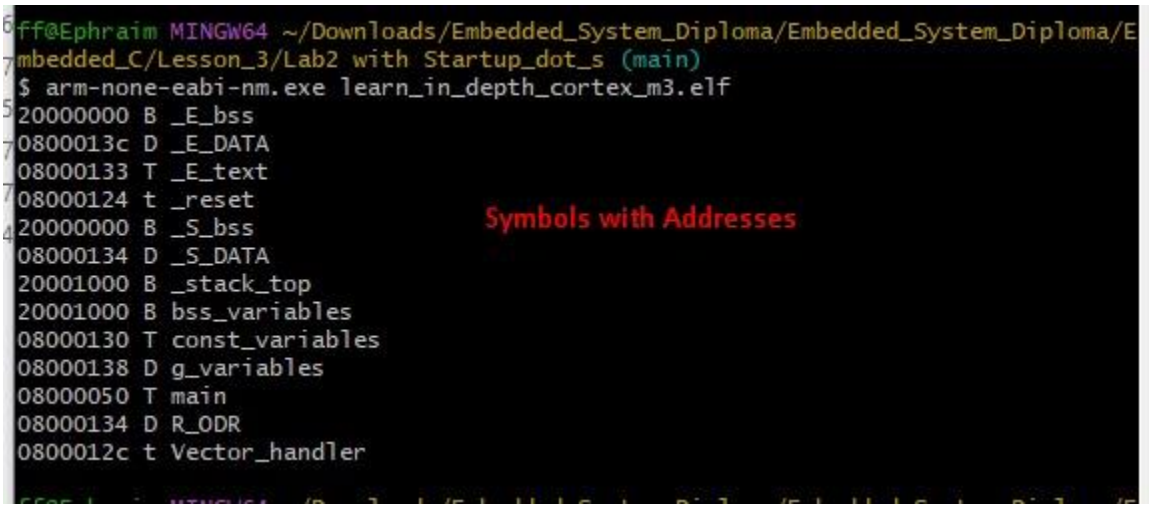
- Use Objdump Command to See Main file Sections



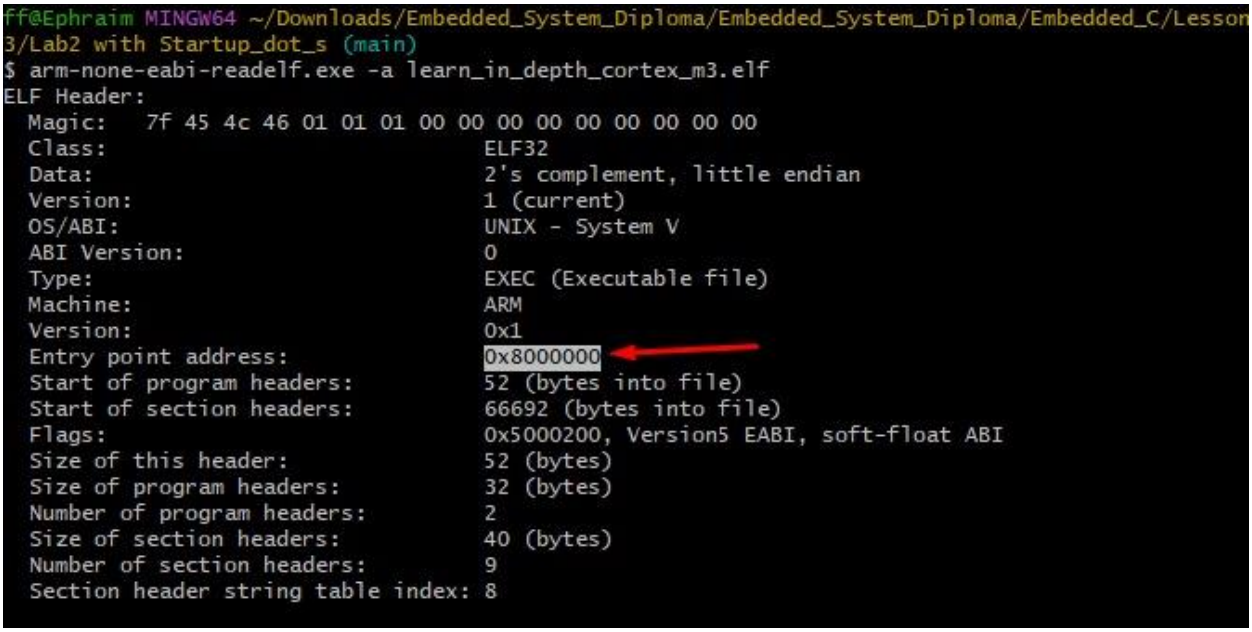
- Use Objdump Command to See elf file Sections



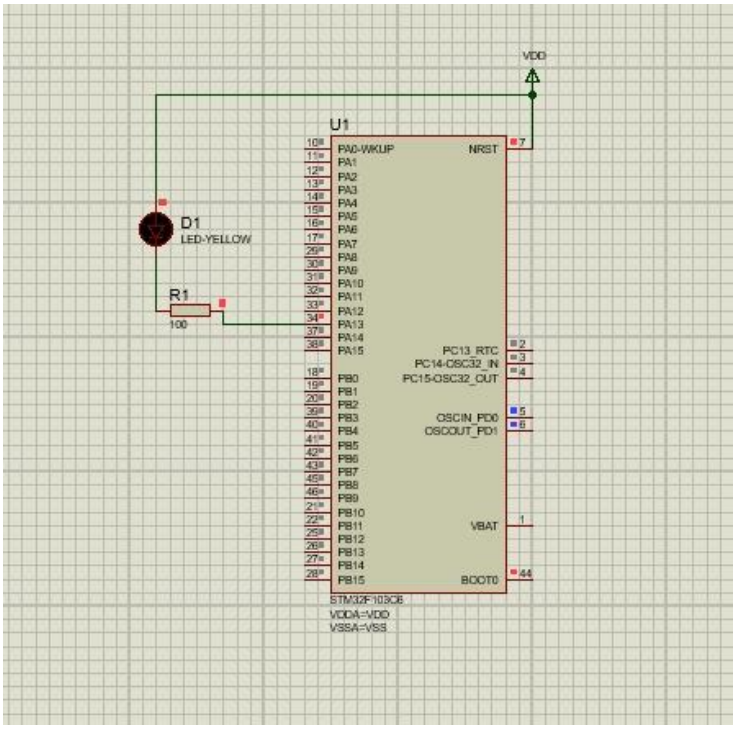
- Use nm Command To See Symbols Details in Elf file

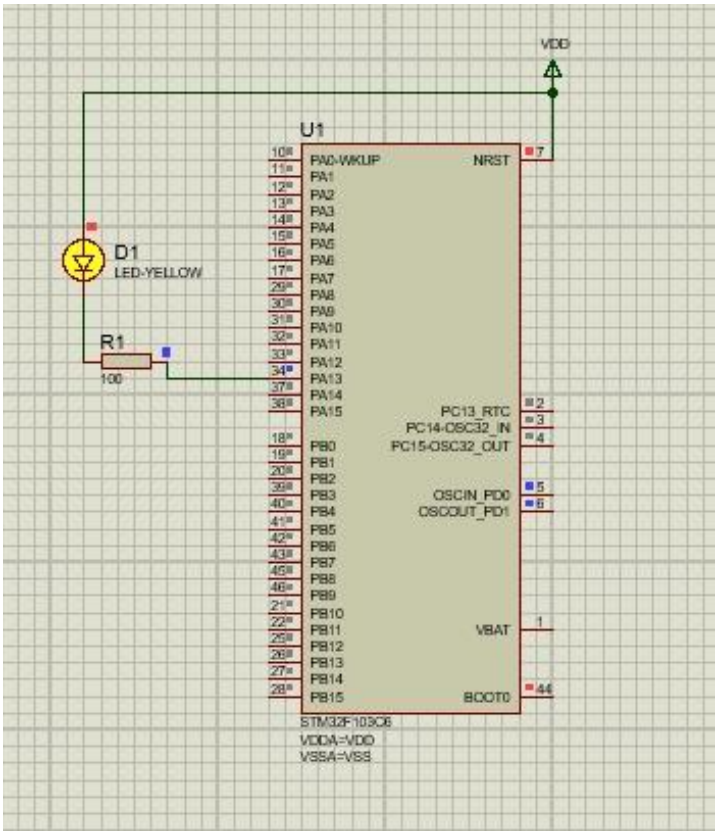


- Use Readelf Command To Check About The Entry Point



- Output in Proteus





Lesson 3 Lab 3 with Startup.c file :

- Main.c file

main.c

```c
typedef volatile unsigned int vint32_t;

//Registers Address
#define RCC_BASE     0x40021000
#define GPIOA_BASE   0x40010800
#define RCC_APB2ENR  *(volatile uint32_t *) (RCC_BASE+0x18)
#define GPIOA_CRH    *(volatile uint32_t *) (GPIOA_BASE+0x04)
#define GPIOA_ODR    *(volatile uint32_t *) (GPIOA_BASE+0x0C)
//Bit Fields
#define RCC_IOPAEN  (1<<2)
#define GPIOA13     (1UL<<13)


typedef union {
        vint32_t    all_fields;
        struct {
        vint32_t    reserved:13 ;
        vint32_t    pin13:1 ;
        }pin;
}R_ODR_t;

volatile R_ODR_t* R_ODR=(volatile R_ODR_t*)(GPIOA_BASE+0x0C);
unsigned char g_variables[3]={1,2,3};
unsigned char const const_variables[3]={1,2,3};
unsigned char volatile bss_Variable[3];
int main(void)
{
    RCC_APB2ENR |=RCC_IOPAEN;
    GPIOA_CRH &=0xFF0FFFFF  ;
    GPIOA_CRH |=0x00200000 ;
    while(1)
    {
//      GPIO_ODR |=(1<<13); //Set Bit 13
        R_ODR->pin.pin13=1;
        for (int i=0 ;i<5000;i++);

//      GPIO_ODR &=~(1<<13); //Clear Bit 13
        R_ODR->pin.pin13=0;
        for (int i=0 ;i<5000;i++);

    }

    return 0;
```

- Startup.c File

```c
#include <stdint.h>
#define STACK_Start_SP 0x20001000
extern int main (void);

void Reset_Handler(void) ;

void Default_Handler()
{
    Reset_Handler();
}

void NMI_Handler (void) __attribute__ ((weak, alias ("Default_Handler"))) ;
void H_Fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;

void MM_Fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;

void Bus_Fault(void) __attribute__ ((weak, alias ("Default_Handler")));;

void Usage_Fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;

extern unsigned int _stack_top;
uint32_t vectors[] __attribute__((section(".vectors")))={

(uint32_t)  &_stack_top,
(uint32_t)  &Reset_Handler,
(uint32_t)  &NMI_Handler,
(uint32_t)  &H_Fault_Handler,
(uint32_t)  &MM_Fault_Handler,
(uint32_t)  &Bus_Fault,
(uint32_t)  &Usage_Fault_Handler

};
```

To Write A Command to Compiler

This Command To Can rewrite The Defination of This Function in Onther File

This Command to make The Function Refer to The address of Reset_Handler function and will be has its Special Address if it has A Defination

This Functions To Initialize The Content of Vector Table

This Command To Make this Section Output in .vectors Section

```c
extern unsigned int _E_text ;
extern unsigned int _S_DATA ;
extern unsigned int _E_DATA ;
extern unsigned int _S_bss ;
extern unsigned int _E_bss ;


void Reset_Handler(void)
{
    //copy data Section From Flash to Ram
    unsigned int DATA_size =(unsigned char*) &_E_DATA - (unsigned char*)&_S_DATA ;//
    unsigned char* P_src =(unsigned char*)&_E_text;
    unsigned char *P_dst =(unsigned char*)&_S_DATA;

    for(int i=0;i<DATA_size;i++)
    {
        *((unsigned char *)P_dst++) = *((unsigned char *)P_src++) ;
    }
    //init .bss section in SRAM =0
    unsigned int bss_size =(unsigned char*) &_E_bss - (unsigned char*)&_S_bss ;
    P_dst=(unsigned char*)&_S_bss;
    for(int i=0 ;i<bss_size;i++)
    {
        *((unsigned char *)P_dst++) = (unsigned char)0 ;
    }
    //jump main()
    main();
}
```

Calculate The Size of .data Section and copy it From flash to SRAM

Calculate The Size of .bss Section And initialize it with 0 in SRAM memory

Branch to The main

- LinkerScript File

```
/*Linker Script CortexM3
Guirguis Hedia
*/
MEMORY
{
flash(RX) : ORIGIN =0x08000000, LENGTH =128K
sram(RWX) : ORIGIN =0x20000000, LENGTH =20K

}

SECTIONS
{
    .text : {
            *(.vectors*)
            *(.text*)
            *(.rodata)
             _E_text = .;
    }>flash

    .data : {
    _S_DATA = . ;
    *(.data)
    . = ALIGN(4) ;
    _E_DATA = . ;

    }>sram AT> flash

    .bss : {
    _S_bss = . ;
    *(.bss*)
    _E_bss = . ;
    . = ALIGN(4);
    . = . +0x1000;
    _stack_top = .;

    }> sram
}
```

Flash Memory Begin with Address 0x080000000 and its size =128k bytes

SRAM Memory Begin with Address 0x200000000 and its size =20k bytes

.text Section Will Be in Flash which Address begin with 0x080000000

.data Section will be in Flash which address =0x080000000 at Burn Time But Moves to SRAM in Execution Time which Address =0x200000000

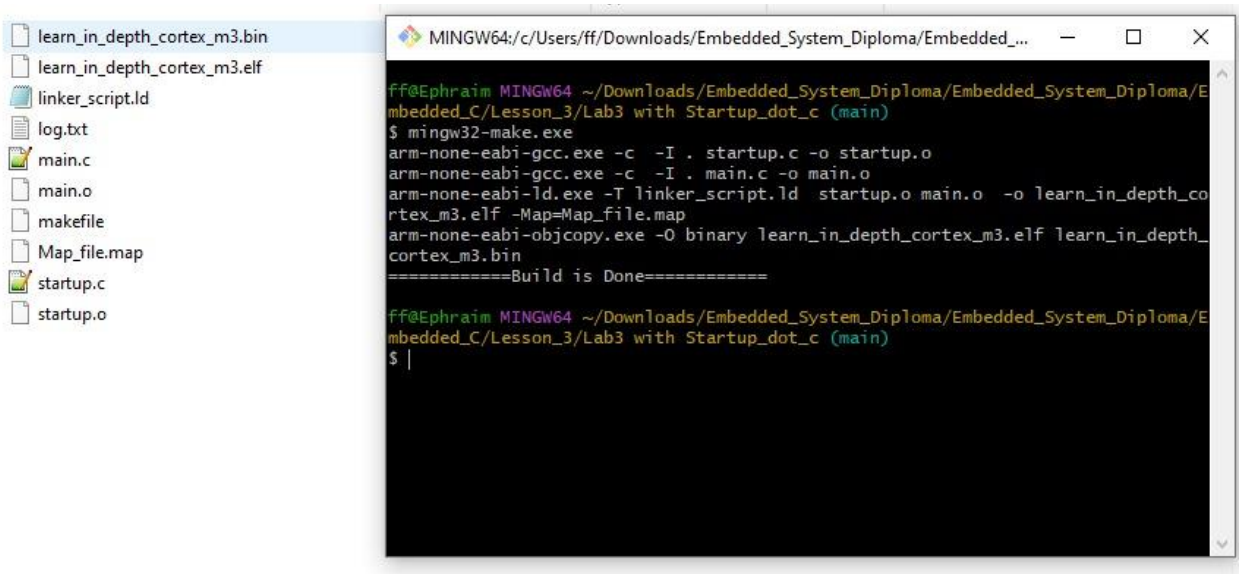
This Command To Make The Memry Alignment

Determine The Size of Stack with 4k Byte

- Make File

```
#@Copyright : Gulguls Media

CC=arm-none-eabi-
CFLAG=-gdwarf-2 -mcpu=cortex-m3
INCS=-I .
LIBS=
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
As= $(wildcard *.s)
AsOBJ= $(As:.s=.o)

Project_Name=learn_in_depth_cortex_m3

all: $(Project_Name).bin
    @echo "============Build is Done============"
#startup.o: startup.s
#    $(CC)as.exe $(CFLAGS) $< -o $@

%.o: %.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@


$(Project_Name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsOBJ) -o $@ -Map=Map_file.map

$(Project_Name).bin: $(Project_Name).elf

    $(CC)objcopy.exe -O binary $< $@

clean_all:
        rm *.o *.elf *.bin

clean:
        rm *.elf *.bin
```

- Build is Done



- Map file Details

```
Allocating common symbols
Common symbol       size              file

bss_Variable        0x3               main.o

Memory Configuration

Name             Origin             Length             Attributes
flash            0x08000000         0x00020000         xr
sram             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map

.text           0x08000000        0x1ff
 *(.vectors*)
 .vectors       0x08000000        0x1c startup.o
                0x08000000                vectors
 *(.text*)
 .text          0x0800001c        0x10c startup.o
                0x0800001c                H_Fault_Handler
                0x0800001c                MM_Fault_Handler
                0x0800001c                Usage_Fault_Handler
                0x0800001c                Bus_Fault
                0x0800001c                Default_Handler
                0x0800001c                NMI_Handler
                0x08000038                Reset_Handler
 .text          0x08000128        0xd4 main.o
                0x08000128                main
 *(.rodata)
 .rodata        0x080001fc        0x3 main.o
                0x080001fc                const_variables
                0x080001ff                _E_text = .

.glue_7         0x08000200        0x0
 .glue_7        0x08000200        0x0 linker stubs

.glue_7t        0x08000200        0x0
 .glue_7t       0x08000200        0x0 linker stubs

.vfp11_veneer   0x08000200        0x0
```

```
 .rel.iplt      0x08000200        0x0 startup.o

.data           0x20000000        0x8 load address 0x080001ff
                0x20000000                _S_DATA = .
 *(.data)
 .data          0x20000000        0x0 startup.o
 .data          0x20000000        0x7 main.o
                0x20000000                R_ODR
                0x20000004                g_variables
                0x20000008                . = ALIGN (0x4)
 *fill*         0x20000007        0x1
                0x20000008                _E_DATA = .

.igot.plt       0x20000008        0x0 load address 0x08000207
 .igot.plt      0x20000008        0x0 startup.o

.bss            0x20000008     0x1003 load address 0x08000207
                0x20000008                _S_bss = .
 *(.bss*)
 .bss           0x20000008        0x0 startup.o
 .bss           0x20000008        0x0 main.o
                0x20000008                _E_bss = .
                0x20000008                . = ALIGN (0x4)
                0x20001008                . = (. + 0x1000)
 *fill*         0x20000008     0x1000
                0x20001008                _stack_top = .
 COMMON         0x20001008        0x3 main.o
                0x20001008                bss_Variable
LOAD startup.o
LOAD main.o
OUTPUT(learn_in_depth_cortex_m3.elf elf32-littlearm)

.comment        0x00000000       0x7e
 .comment       0x00000000       0x7e startup.o
                                 0x7f (size before relaxing)
 .comment       0x0000007e       0x7f main.o
```

.data Section Here in SRAM with Address 0x20000000

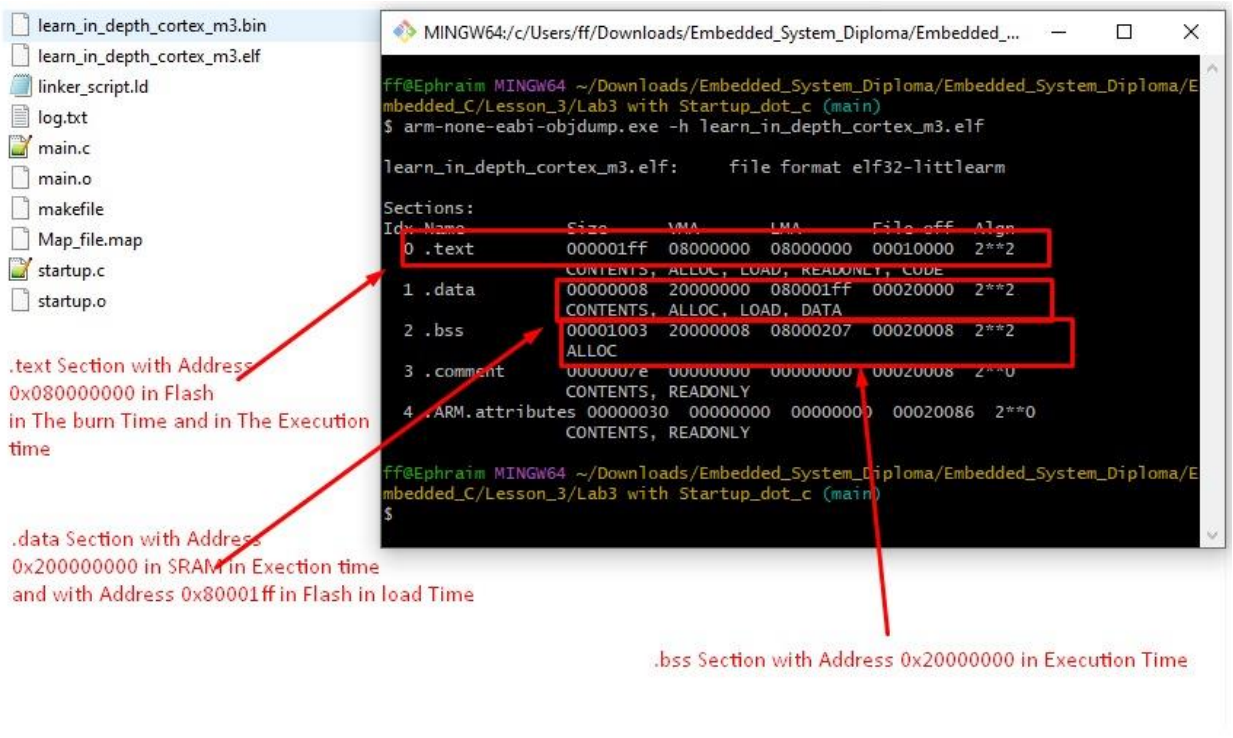
.bss Section Begin with Address 0x20000008 in The SRAM Memory

Stack_Top with Address 0x20001008
Which Means The Stack Memory with Size 0x1000 "4k Byte"

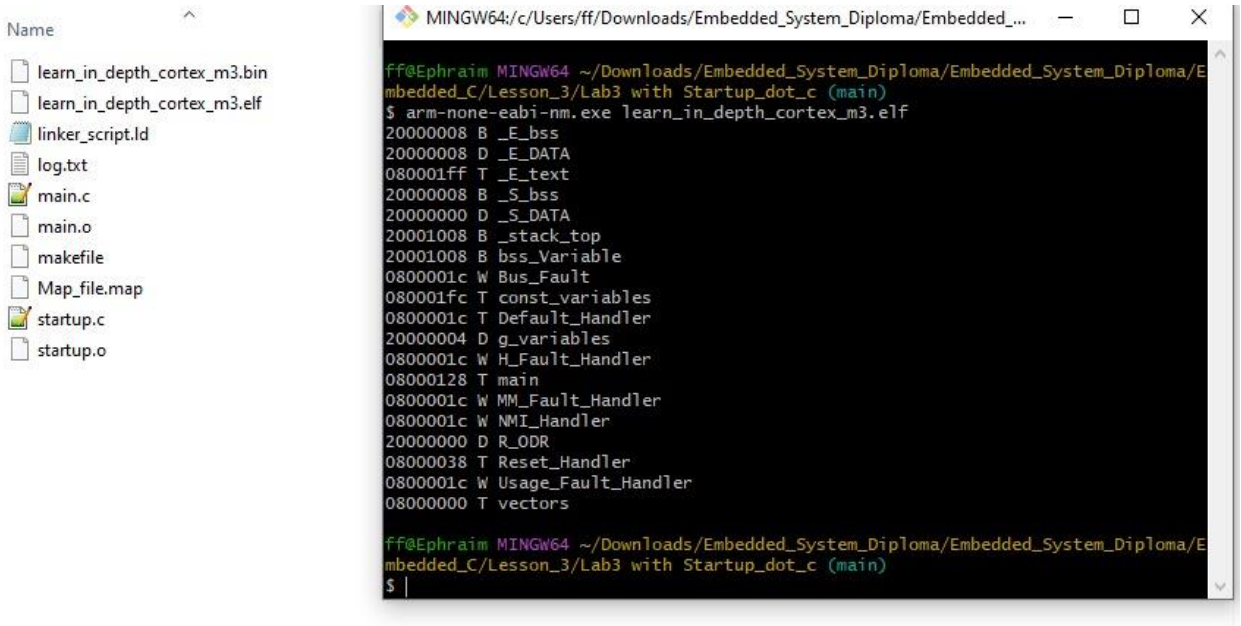
- Use Command Objdump To For main.o File

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

main.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000d4  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000007  00000000  00000000  00000108  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  0000010f  2**0
                  ALLOC
  3 .rodata       00000003  00000000  00000000  00000110  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      0000007f  00000000  00000000  00000113  2**0
                  CONTENTS, READONLY
  5 .ARM.attributes 00000030  00000000  00000000  00000192  2**0
                  CONTENTS, READONLY
```

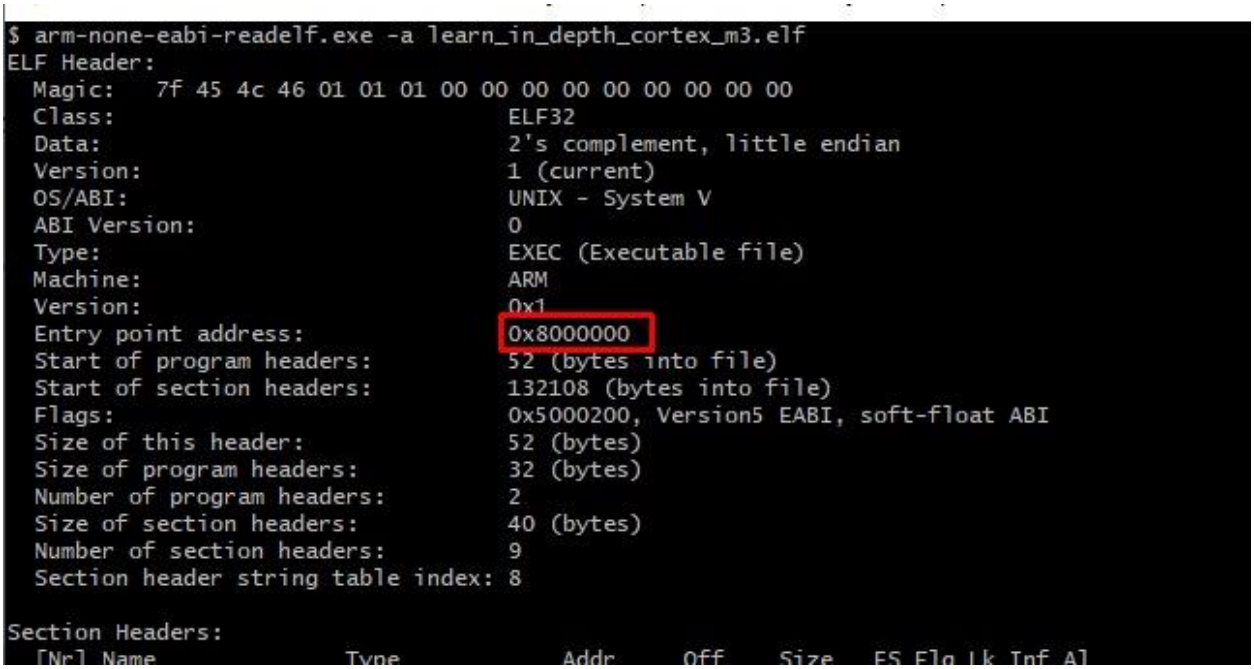
- Use Command Objdump To For The output Elf file



- Use Command nm to See The Symbols of Project elf



- Use Command Readelf



- Output In Proteus