Lap_1 Report

Writing code :

Uart.c

```c
#include <Uart.h>

#define UART0DR *((volatile unsigned int*)((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++;
    }
}
```

Uart.h

```c
#ifndef _UART_H_
#define _UART_H_

void UART_Send_String(unsigned char* P_tx_string);

#endif
```

App.c

```c
#include "Uart.h"

unsigned char string_name[100]= "Ahmed AL-Qasabi" ;
unsigned char const string_name_2[100]= "Ahmed AL-Qasabi" ;
void main(){
    UART_Send_String(string_name);
}
```

Startup.s

```asm
.global reset

reset:
    ld sp, =stack_top
    bl main

stop:   b stop
```

Lunker_Script.ld

```ld
ENTRY(reset)

MEMORY{
    Mem (rwx):ORIGIN = 0X00000000 , LENGTH = 64M
}

SECTION{

    . = 0X10000;
    .ahmed_startup . :
    {
        startup.o(.text)
    }> Mem

    .ahmed_text_rodata :
    {
        *(.text) *(.rodata)
    }> Me

    .ahmed_data :
    {
        *(.data)
    }> Mem

    .ahmed_bss :
    {
        *(.bss)
    }> Mem

    .= . +0x1000;
    stack_top = . ;
}
```

Getting object files With debug informations and sections :

Getting object files with debug informations

Uart.o Sections

App.o Sections

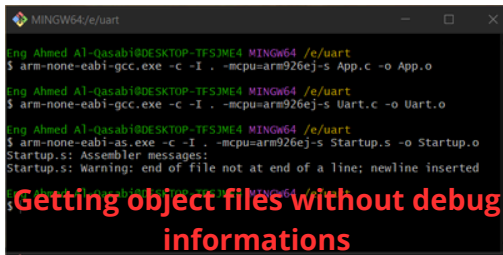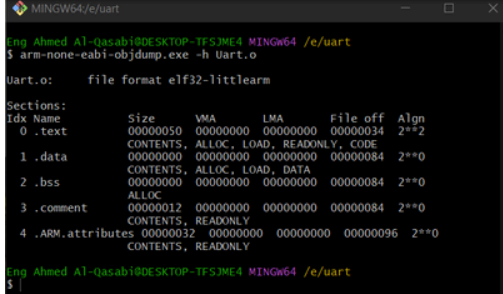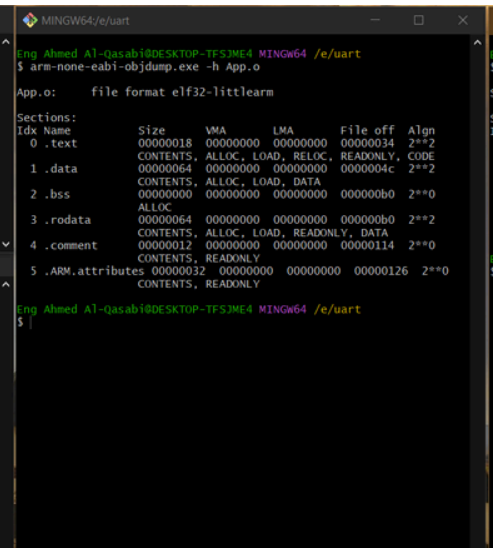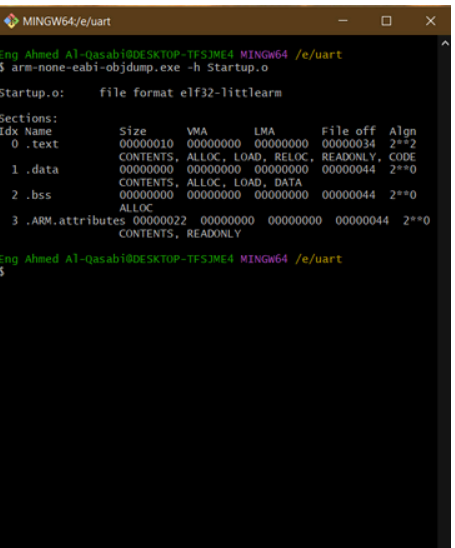Startup.o Sections

Getting object files Without debug informations and sections:

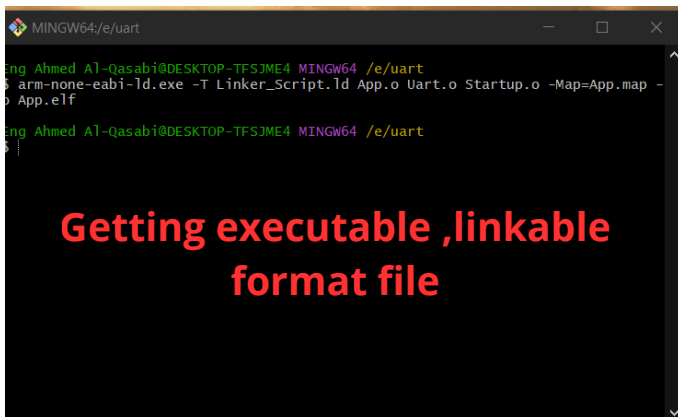**Getting object files without debug informations**
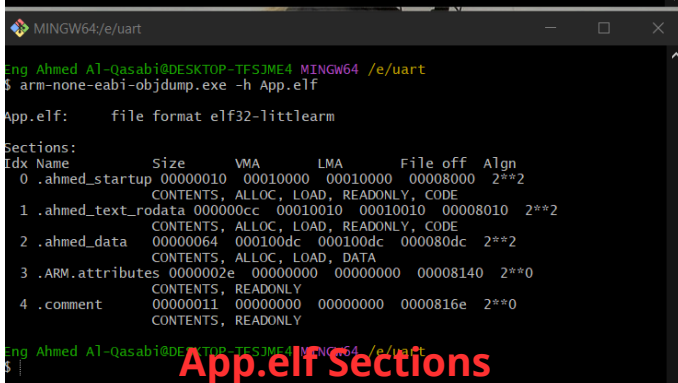
Uart.o Sections

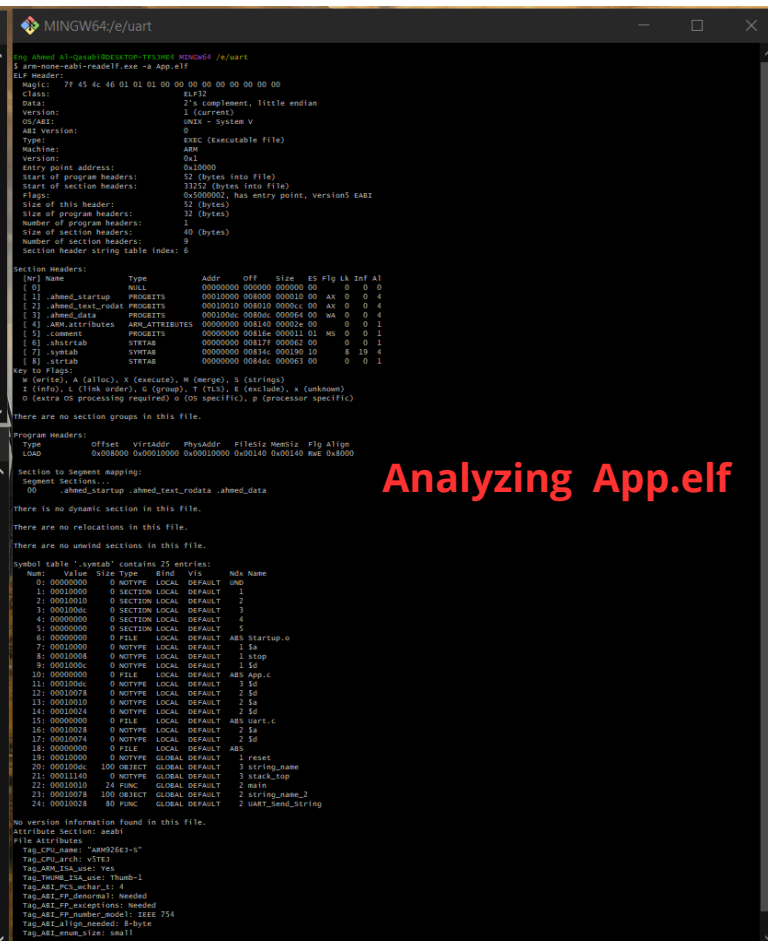**App.o Sections**

Startup.o Sections

Getting executable ,linkable format file and sections :

Getting executable ,linkable format file

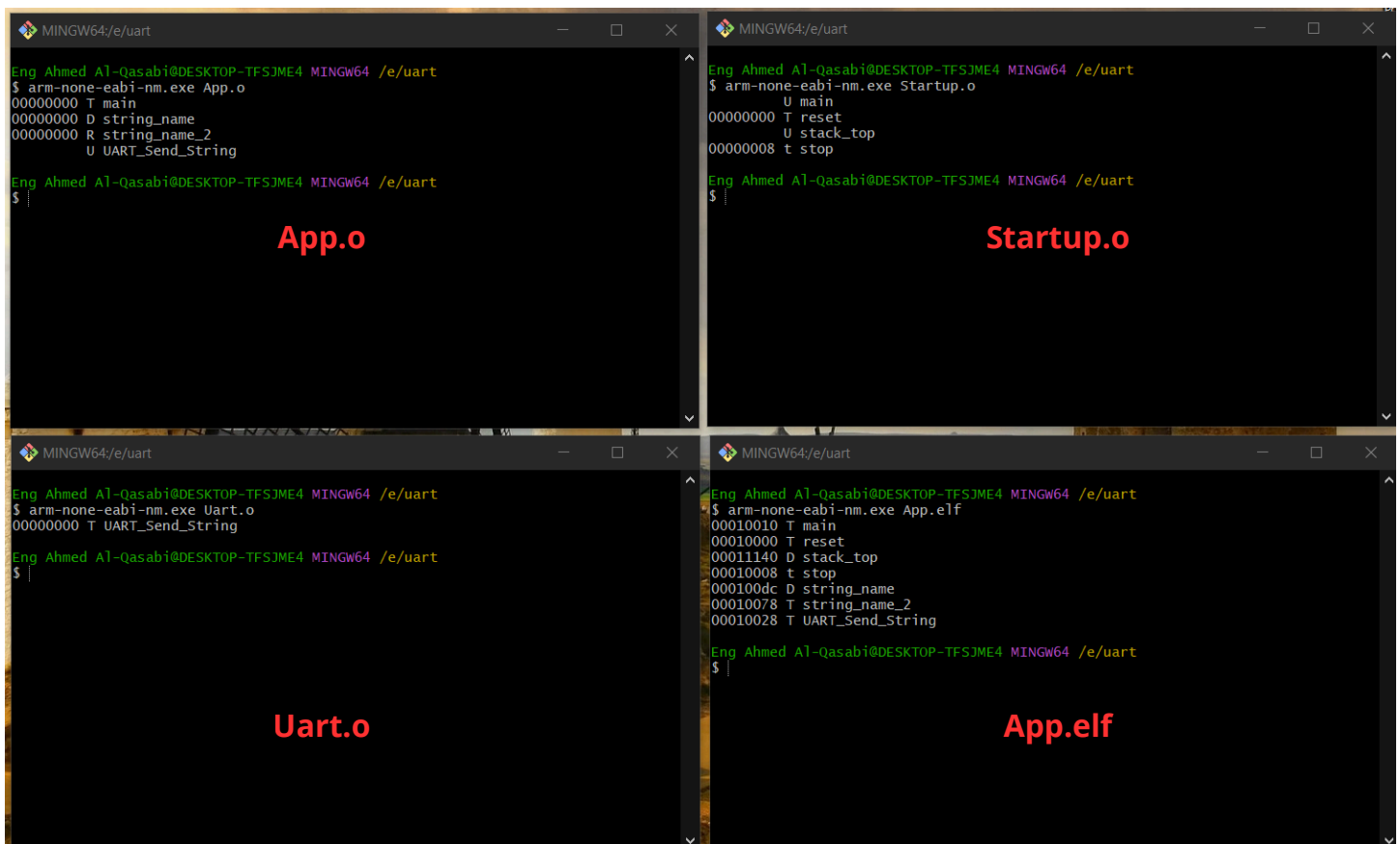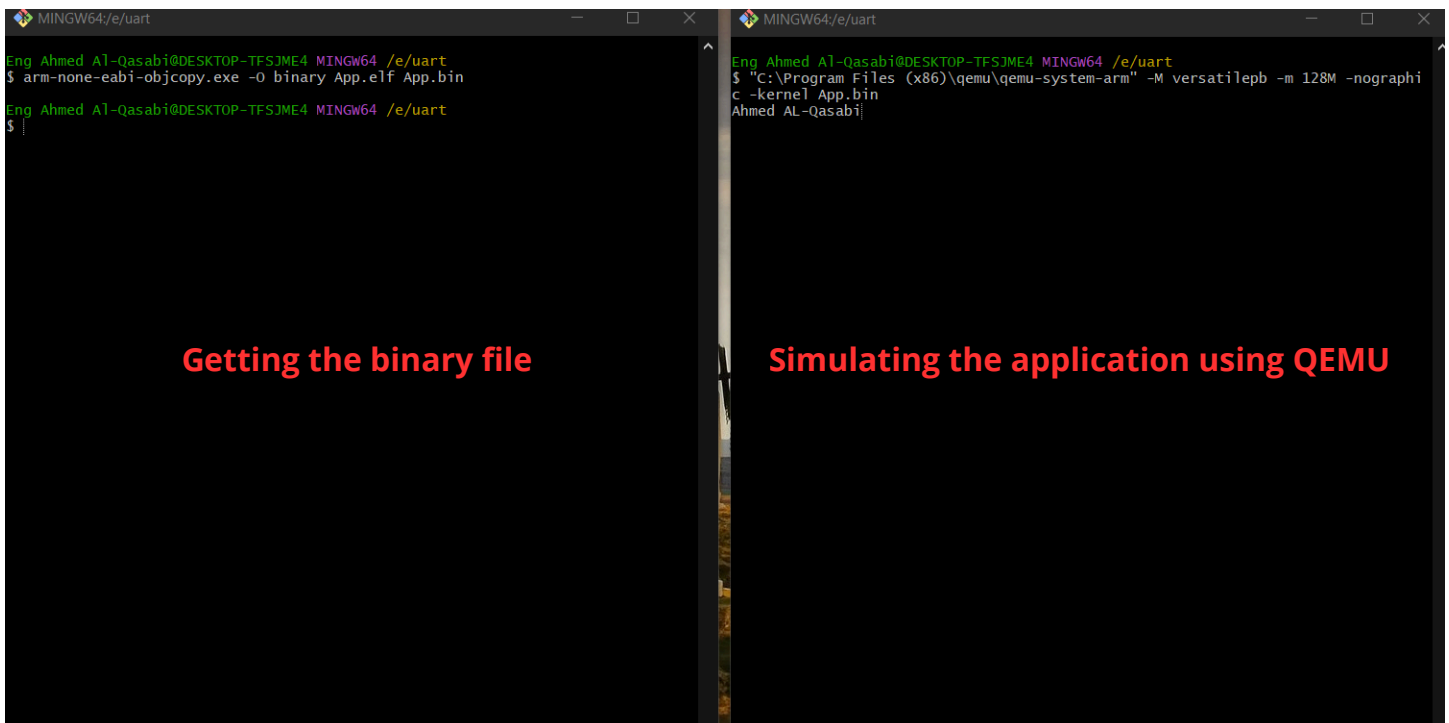**App.elf Sections**

Analyzing App.elf

Getting the symbol table for the object files and the final elf file

```
MINGW64:/e/uart                                    —  □  ×

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$ arm-none-eabi-nm.exe App.o
00000000 T main
00000000 D string_name
00000000 R string_name_2
         U UART_Send_String

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$
```
App.o

```
MINGW64:/e/uart                                    —  □  ×

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$ arm-none-eabi-nm.exe Startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$
```
Startup.o

```
MINGW64:/e/uart                                    —  □  ×

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$ arm-none-eabi-nm.exe Uart.o
00000000 T UART_Send_String

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$
```
Uart.o

```
MINGW64:/e/uart                                    —  □  ×

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$ arm-none-eabi-nm.exe App.elf
00010010 T main
00010000 T reset
00011140 D stack_top
00010008 t stop
000100dc D string_name
00010078 T string_name_2
00010028 T UART_Send_String

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$
```
App.elf

Getting the binary file and simulating the application using QEMU

```
MINGW64:/e/uart                                    —  □  ×

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$ arm-none-eabi-objcopy.exe -O binary App.elf App.bin

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$
```
Getting the binary file

```
MINGW64:/e/uart                                    —  □  ×

Eng Ahmed Al-Qasabi@DESKTOP-TFSJME4 MINGW64 /e/uart
$ "C:\Program Files (x86)\qemu\qemu-system-arm" -M versatilepb -m 128M -nographi
c -kernel App.bin
Ahmed AL-Qasabi
```
Simulating the application using QEMU