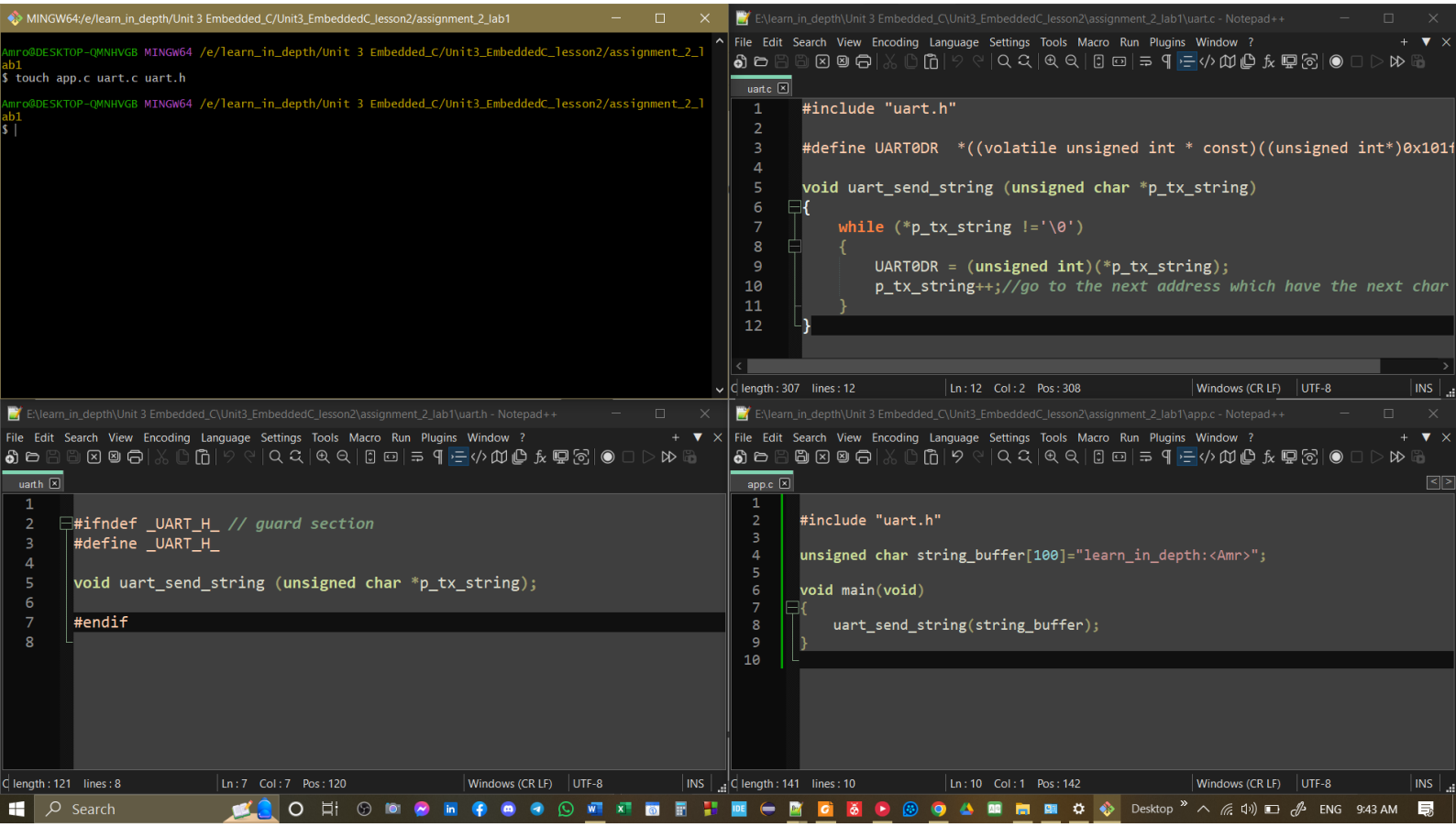
Lab1

- Creating from scratch a baremetal software to send a string using UART0
- We will write c_code, linker_script and startup_code and use the binary utilities
- The whole code will written using only arm-none-eabi toolchain without using any IDE

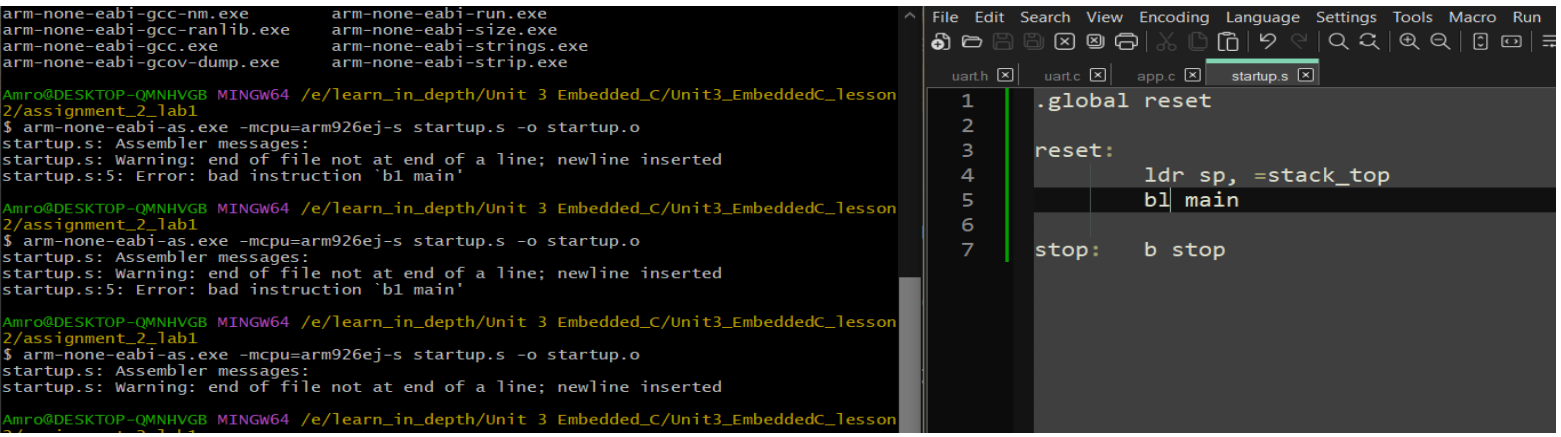
C_codes



To generate object files (app.o and uart.o)

```
arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I . app.c -o app.o
arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I . uart.c -o uart.o
```

next step is the Assymbler , so we have to make startup.s



Binary utilities of different stages of the code.

```
Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/U
_lesson2/assignment_2_lab1
$ 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

Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/U
2/assignment_2_lab1
$ 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

Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/U
2/assignment_2_lab1
$ 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

Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/U
2/assignment_2_lab1
$
```

Note that all addresses are not physical (is virtual addresses) that because they are object file and they will resolved and allocated in the linker stage specifically with linker_script.ld

```
Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/Unit3_Embedde
_lesson2/assignment_2_lab1
$ arm-none-eabi-objdump.exe -D app.o

app.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0:   e92d4800        push    {fp, lr}
   4:   e28db004        add     fp, sp, #4
   8:   e59f0004        ldr     r0, [pc, #4]    ; 14 <main+0x14>
   c:   ebfffffe        bl      0 <uart_send_string>
  10:   e8bd8800        pop     {fp, pc}
  14:   00000000        andeq   r0, r0, r0

Disassembly of section .data:

00000000 <string_buffer>:
   0:   7261656c        rsbvc   r6, r1, #108, 10        ; 0x1b000000
   4:   6e695f6e        cdpvs   15, 6, cr5, cr9, cr14, {3}
   8:   7065645f        rsbvc   r6, r5, pc, asr r4
   c:   3c3a6874        ldccc   8, cr6, [sl], #-464     ; 0xfffffe30
  10:   3e726d41        cdpcc   13, 7, cr6, cr2, cr1, {2}
        ...

Disassembly of section .comment:

00000000 <.comment>:
   0:   43434700        movtmi  r4, #14080      ; 0x3700
   4:   4728203a                        ; <UNDEFINED> instruction: 0x4728203a
   8:   2029554e        eorcs   r5, r9, lr, asr #10
   c:   2e372e34        mrccs   14, 1, r2, cr7, cr4, {1}
  10:   Address 0x00000010 is out of bounds.


Disassembly of section .ARM.attributes:

00000000 <.ARM.attributes>:
   0:   00003141        andeq   r3, r0, r1, asr #2
   4:   61656100        cmnvs   r5, r0, lsl #2
   8:   01006962        tsteq   r0, r2, ror #18
   c:   00000027        andeq   r0, r0, r7, lsr #32
  10:   4d524105        ldfmie  f4, [r2, #-20]  ; 0xfffffec
  14:   45363239        ldrmi   r3, [r6, #-569]!        ; 0x239
  18:   00532d4a        subseq  r2, r3, sl, asr #26
  1c:   01080506        tsteq   r8, r6, lsl #10
  20:   04120109        ldreq   r0, [r2], #-265 ; 0x109
  24:   01150114        tsteq   r5, r4, lsl r1
  28:   01180317        tsteq   r8, r7, lsl r3
  2c:   011a0119        tsteq   sl, r9, lsl r1
  30:   Address 0x00000030 is out of bounds.


Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/Unit3_Embedde
_lesson2/assignment_2_lab1
```

Next step is the linker , so we have to make linker_script.ld to control all memory locations and sizes , starting point and stack size

uart.h | uart.c | app.c | startup.s | linker_script.ld

```
ENTRY(reset)
MEMORY
{
    Mem(rwx):ORIGIN = 0x00000000 , LENGTH = 64M
}
SECTIONS
{
    . =0x10000;
    .startup_afifi . :
    {
        startup.o (.text)
    }>Mem

    .text_afifi :
    {
        *(.text) *(.rodata)
    }>Mem

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

    .bss_afifi :
    {
        *(.bss) *(COMMON)
    }>Mem
    . = . + 0x1000;
    stack_top = . ;
}
```

Then liking all object files (app.o , uart.o , startup.o)and linker_script.ld to generate the .elf and .map files

```
arm-none-eabi-ld.exe -T linker_script.ld -Map=map_file.map app.o uart.o startup.o -o learn_in_depth.elf
```

After that generating the binary code that will burnt on the board

```
$ arm-none-eabi-objcopy.exe -O binary learn_in_depth.elf learn_in_depth.bin
```

Note that symbols have been resolved and all sections allocated on the physical addresses which identified in linker_script

```
Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/Unit3_Embedded_C_lesson 2/assignment_2_lab1
$ arm-none-eabi-objdump.exe -h learn_in_depth.elf

learn_in_depth.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .startup_afifi 00000010  00010000  00010000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text_afifi   00000068  00010010  00010010  00008010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data_afifi   00000064  00010078  00010078  00008078  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  000080dc  2**0
                  CONTENTS, READONLY
  4 .comment      00000011  00000000  00000000  0000810a  2**0
                  CONTENTS, READONLY

Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/Unit3_Embedded_C_lesson 2/assignment_2_lab1
$
```

Then call qemu emulator to run the code and see the output

```
Amro@DESKTOP-QMNHVGB MINGW64 /e/learn_in_depth/Unit 3 Embedded_C/Unit3_EmbeddedC_lesson 2/assignment_2_lab1
$  qemu-system-arm.exe -M versatilepb -m 128M -nographic -kernel learn_in_depth.bin
learn_in_depth:<Amr>
```