

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:

```
MINGW32:/c/ARM_TOOLCHAIN/bin
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-gcc.exe -c -I. -g -mcpu=arm926ej-s app.c -o app.o

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-gcc.exe -c -I. -g -mcpu=arm926ej-s uart.c -o uart.o

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

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

```
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 .debug_info     0000006b  00000000  00000000  000000b0  2**0
    CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev   00000058  00000000  00000000  0000011b  2**0
    CONTENTS, READONLY, DEBUGGING
  5 .debug_loc      0000002c  00000000  00000000  00000173  2**0
    CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges  00000020  00000000  00000000  0000019f  2**0
    CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line     00000035  00000000  00000000  000001bf  2**0
    CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str      00000051  00000000  00000000  000001f4  2**0
    CONTENTS, READONLY, DEBUGGING
  9 .comment        00000012  00000000  00000000  00000245  2**0
    CONTENTS, READONLY
10 .ARM.attributes  00000032  00000000  00000000  00000257  2**0
    CONTENTS, READONLY
11 .debug_frame     0000002c  00000000  00000000  0000028c  2**2
    CONTENTS, RELOC, READONLY, DEBUGGING

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

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 startup.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:

MINGW32:/c/ARM_TOOLCHAIN/bin

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

app.elf:          file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .startup       0000000c  00010000  00010000  00008000  2**2
    CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text          00000068  0001000c  0001000c  0000800c  2**2
    CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data          00000064  00010074  00010074  00008074  2**2
    CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  000080d8  2**0
    CONTENTS, READONLY
  4 .comment        00000011  00000000  00000000  00008106  2**0
    CONTENTS, READONLY

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

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

MINGW32:/c/ARM_TOOLCHAIN/bin

```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-nm.exe app.elf
00010010 T main
00010000 T reset
000110dc D stack_top
00010008 t stop
00010078 D string_buffer
00010028 T Uart_send_string

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

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

MINGW32:/c/ARM_TOOLCHAIN/bin

```
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_buffer
          U Uart_send_string

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-nm.exe uart.o
00000000 T Uart_send_string

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-nm.exe startup.o
          U main
00000000 T reset
          U stack_top

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

- 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:

```

MINGW32/c/ARM_TOOLCHAIN/bin
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-objcopy.exe -O binary app.elf app.bin

mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ arm-none-eabi-readelf.exe -a app.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF32
  Data:                             2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:                0x10000
  Start of program headers:          52 (bytes into file)
  Start of section headers:         33120 (bytes into file)
  Flags:                             0x5000002, has entry point, Version5 EABI
  Size of this header:                52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:          1
  Size of section headers:           40 (bytes)
  Number of section headers:          9
  Section header string table index: 6

Section Headers:
 [Nr] Name                Type              Addr      Off      Size    ES Flg Lk Inf Al
 [ 0]                     NULL              00000000  000000  000000  00   0  0  0  0
 [ 1] .startup              PROGBITS          00010000  008000  00000c  00  AX  0  0  4
 [ 2] .text                 PROGBITS          0001000c  00800c  000068  00  AX  0  0  4
 [ 3] .data                 PROGBITS          00010074  008074  000064  00  WA  0  0  4
 [ 4] .ARM.attributes       ARM_ATTRIBUTES    00000000  0080d8  00002e  00   0  0  1
 [ 5] .comment              PROGBITS          00000000  008106  000011  01  MS  0  0  1
 [ 6] .shstrtab             STRTAB            00000000  008117  000049  00   0  0  1
 [ 7] .symtab               SYMTAB            00000000  0082c8  000160  10   8 17  4
 [ 8] .strtab               STRTAB            00000000  008428  000052  00   0  0  1

Key to Flags:
 W (write), A (alloc), X (execute), M (merge), S (strings)
 I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
 O (extra OS processing required) o (OS specific), p (processor specific)

```

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

```

MINGW32/c/ARM_TOOLCHAIN/bin
mosta@Mostafa-PC MINGW32 /c/ARM_TOOLCHAIN/bin
$ ../../qemu/qemu-system-arm -M versatilepb -m 128M -nographic -kernel app.bin
learn-in-depth:Mostafa Mahmoud

```