1/8/2022

Embedded C

Lab (3)

Reham Nady Abd Elmotaal

1-Introduction:

Write a baremetal software to toggle led which connected to GPIO portF pin3 ARM Cortex M4 based TM4C123GH6PM microcontroller. I will build everything from scratch including startup ,linker script and source codes ,and compile them using arm cross tool chain.

2-Source codes:

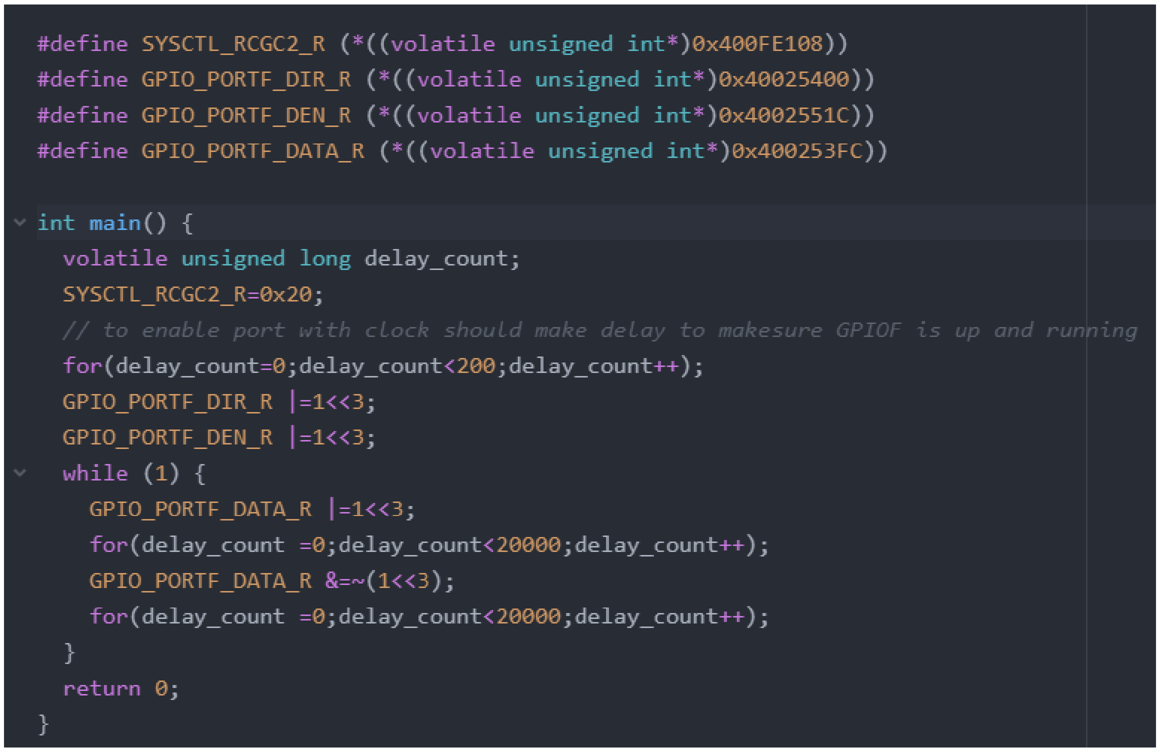
2.1:main.c

To make a GPIO toggling in ARM Cortex4, you need to work with two peripherals:

1-SYSTCL: To Enable the GPIO port.

2- GPIOx (general purpose input/output):

First set(PF3)the direction as output, then enable the GPIO pin, finally toggle bit in GPIO_PORTF_DATA_R.



```
#define SYSCTL_RCGC2_R (*((volatile unsigned int*)0x400FE108))
#define GPIO_PORTF_DIR_R (*((volatile unsigned int*)0x40025400))
#define GPIO_PORTF_DEN_R (*((volatile unsigned int*)0x4002551C))
#define GPIO_PORTF_DATA_R (*((volatile unsigned int*)0x400253FC))

int main() {
  volatile unsigned long delay_count;
  SYSCTL_RCGC2_R=0x20;
  // to enable port with clock should make delay to makesure GPIOF is up and running
  for(delay_count=0;delay_count<200;delay_count++);
  GPIO_PORTF_DIR_R |=1<<3;
  GPIO_PORTF_DEN_R |=1<<3;
  while (1) {
    GPIO_PORTF_DATA_R |=1<<3;
    for(delay_count =0;delay_count<20000;delay_count++);
    GPIO_PORTF_DATA_R &=~(1<<3);
    for(delay_count =0;delay_count<20000;delay_count++);
  }
  return 0;
}
```

3-Startup:

startup written in c code : as (CortexM4) can intalizethe SP with the first 4 bytes, so we can write startup by C code.

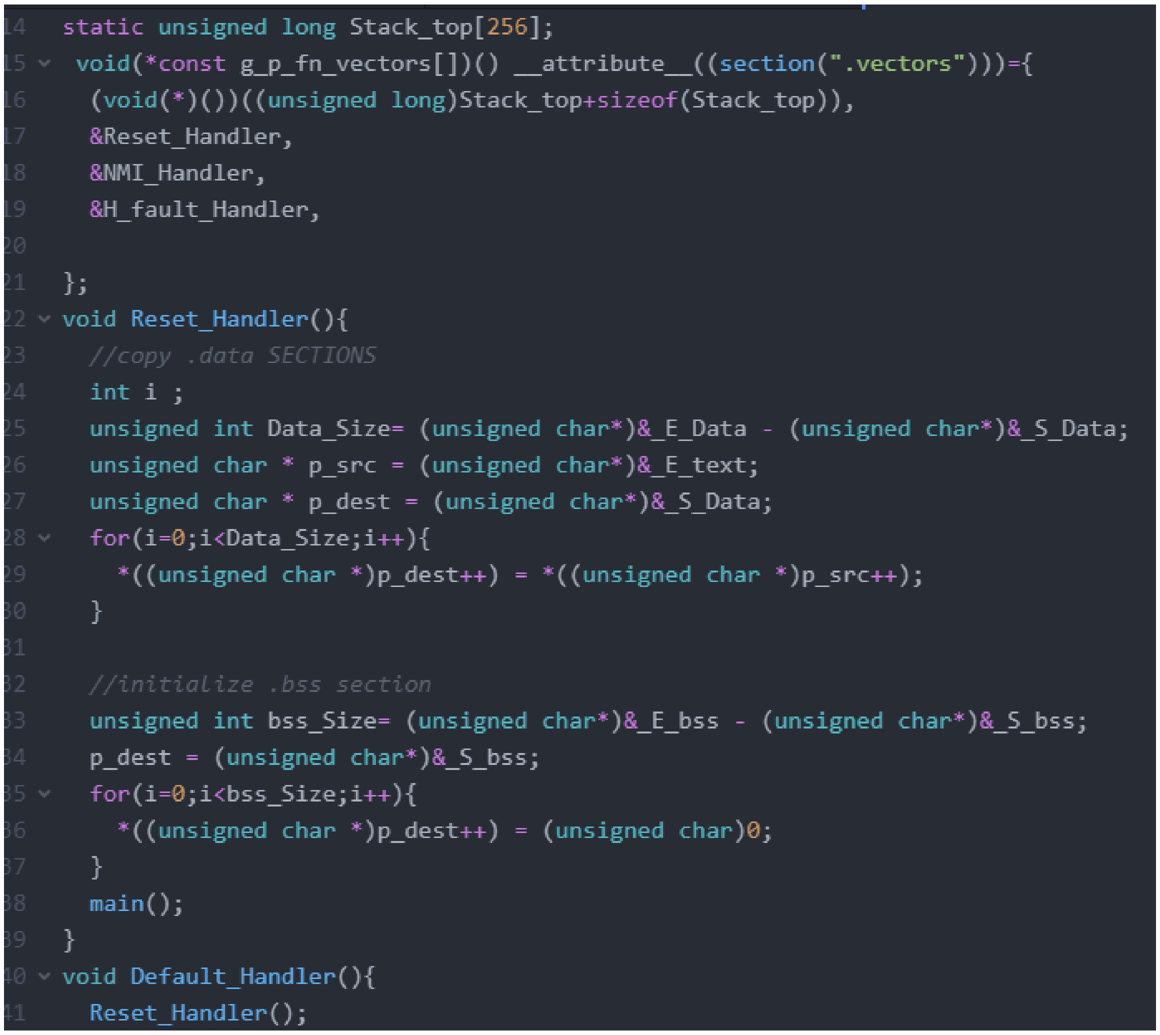
In this code we make :

1-Define Interrupt vectors Section

2-copy .data section from ROM to RAM.

3-initialize .bss section.

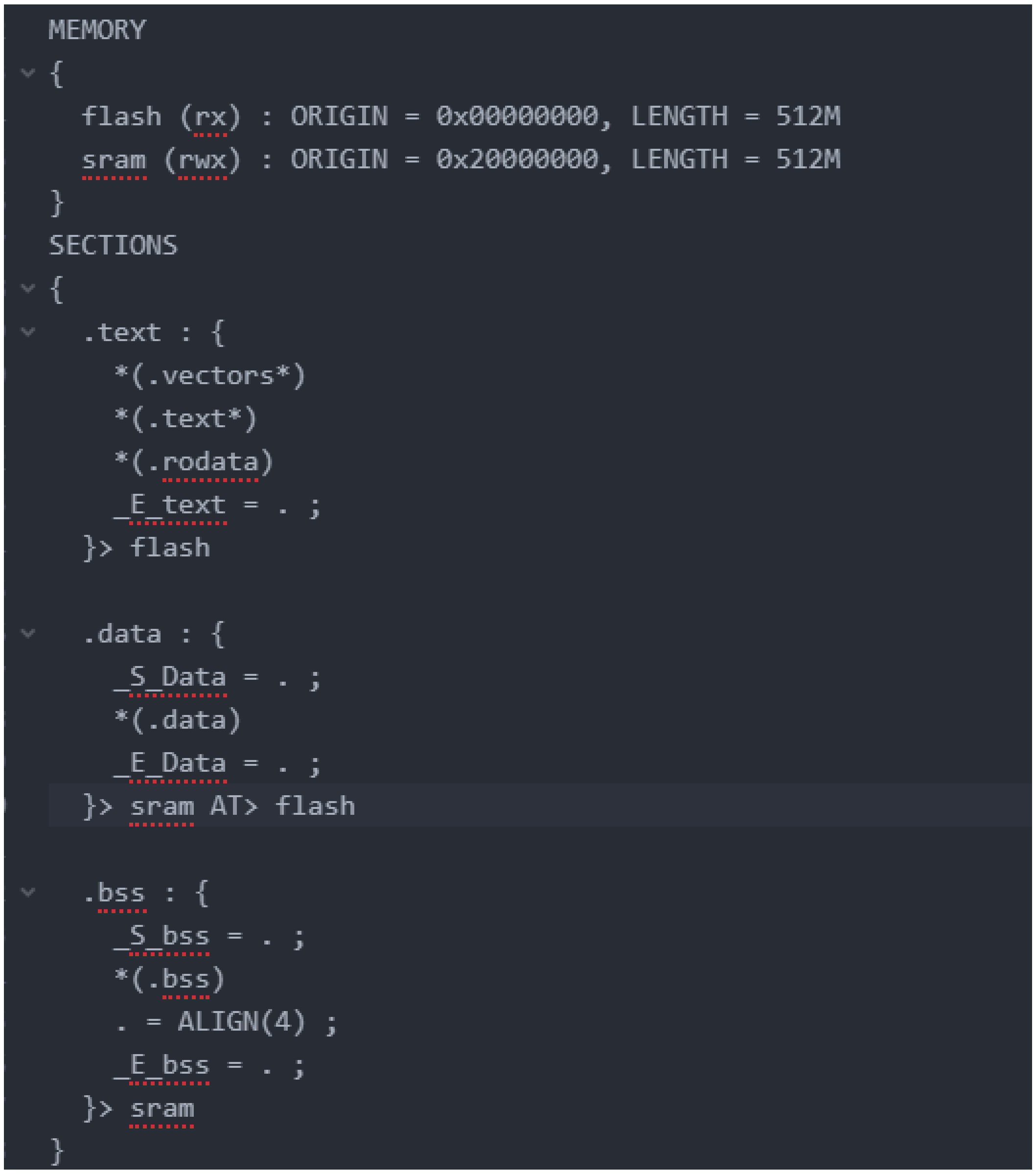
4-Create a reset section and Call main function.



```c
static unsigned long Stack_top[256];
 void(*const g_p_fn_vectors[])() __attribute__((section(".vectors")))={
  (void(*)())((unsigned long)Stack_top+sizeof(Stack_top)),
  &Reset_Handler,
  &NMI_Handler,
  &H_fault_Handler,

};
void Reset_Handler(){
  //copy .data SECTIONS
  int i ;
  unsigned int Data_Size= (unsigned char*)&_E_Data - (unsigned char*)&_S_Data;
  unsigned char * p_src = (unsigned char*)&_E_text;
  unsigned char * p_dest = (unsigned char*)&_S_Data;
  for(i=0;i<Data_Size;i++){
    *((unsigned char *)p_dest++) = *((unsigned char *)p_src++);
  }

  //initialize .bss section
  unsigned int bss_Size= (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
  p_dest = (unsigned char*)&_S_bss;
  for(i=0;i<bss_Size;i++){
    *((unsigned char *)p_dest++) = (unsigned char)0;
  }
  main();
}
void Default_Handler(){
  Reset_Handler();
```

4-Linker script:

In this linker script, we define memory boundaries ,in this app we have just one memory . the last section in linker used to divide my code in all file and organize it to burn it on the micro controller.

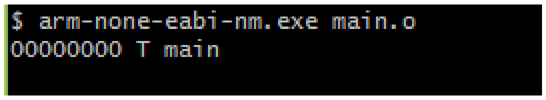


```
MEMORY
{
  flash (rx) : ORIGIN = 0x00000000, LENGTH = 512M
  sram (rwx) : ORIGIN = 0x20000000, LENGTH = 512M
}
SECTIONS
{
  .text : {
    *(.vectors*)
    *(.text*)
    *(.rodata)
    _E_text = . ;
  }> flash

  .data : {
    _S_Data = . ;
    *(.data)
    _E_Data = . ;
  }> sram AT> flash

  .bss : {
    _S_bss = . ;
    *(.bss)
    . = ALIGN(4) ;
    _E_bss = . ;
  }> sram
}
```

5-Symbols:

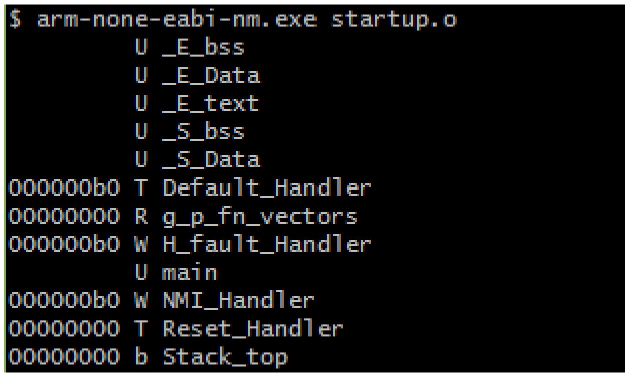
5.1:symbol of main.o:

1- main: which in text section.



```
$ arm-none-eabi-nm.exe main.o
00000000 T main
```

5.3:symbol of startup.o:



```
$ arm-none-eabi-nm.exe startup.o
         U _E_bss
         U _E_Data
         U _E_text
         U _S_bss
         U _S_Data
000000b0 T Default_Handler
00000000 R g_p_fn_vectors
000000b0 W H_fault_Handler
         U main
000000b0 W NMI_Handler
00000000 T Reset_Handler
00000000 b Stack_top
```

1- _E_bss: unresolved symbol and will be resolved during Linking process.

2- _E_Data: unresolved symbol and will be resolved during Linking process.

3- _E_text: unresolved symbol and will be resolved during Linking process.

4- _S_bss: unresolved symbol and will be resolved during Linking process.

5- _S_Data: unresolved symbol and will be resolved during Linking process.

6- Default_handler: which in text section.

7- g_p_fn_habdler: which in read only data section.

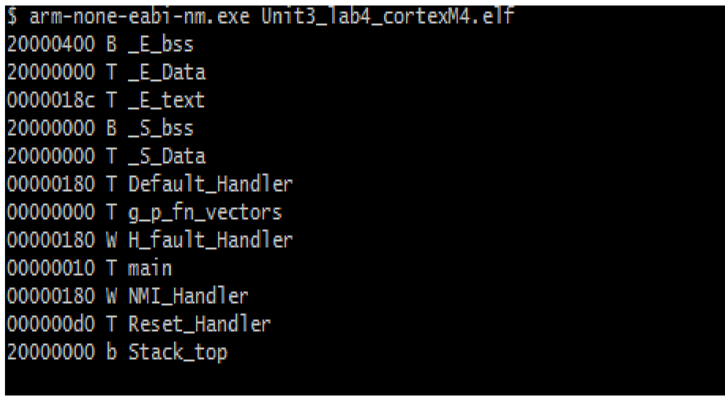
8-H_Fault_handler & NMI_handler: are weak symbol.

9- main: unresolved symbol and will be resolved during Linking process.

10- Reset_handler: which in text section.

11-stack_top: which in bss section.

5.4:elf image sympols:



```
$ arm-none-eabi-nm.exe Unit3_lab4_cortexM4.elf
20000400 B _E_bss
20000000 T _E_Data
0000018c T _E_text
20000000 B _S_bss
20000000 T _S_Data
00000180 T Default_Handler
00000000 T g_p_fn_vectors
00000180 W H_fault_Handler
00000010 T main
00000180 W NMI_Handler
000000d0 T Reset_Handler
20000000 b Stack_top
```

1- _E_bss: which in bss section.

2- _E_Data: which in text section.

3- _E_text: which in text section.

4- _S_bss: which in bss section.

5- _S_Data: which in text section.

6- Default_handler: which in text section.

7- g_p_fn_habdler: which in read only text section.

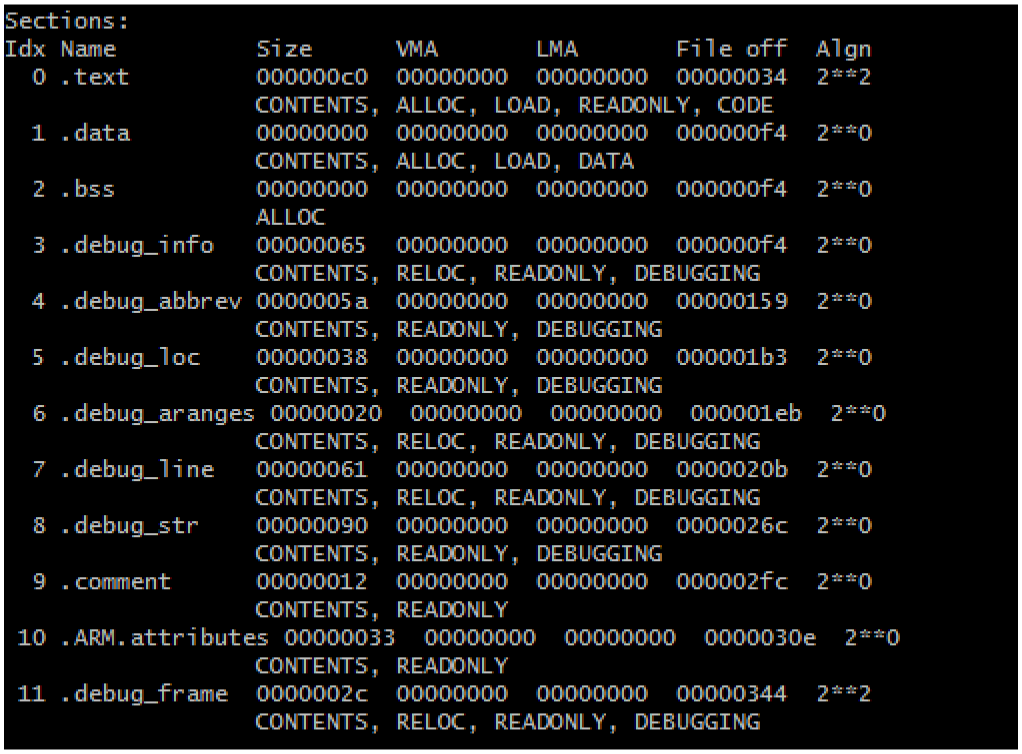
8-H_Fault_handler & NMI_handler: are weak symbol.

9- main: which in text section.

10- Reset_handler: which in text section.

11-stack_top: which in bss section.

6-Sections Headers:
6.1: main.o sections headers



```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000c0  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000f4  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000f4  2**0
                  ALLOC
  3 .debug_info   00000065  00000000  00000000  000000f4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 0000005a  00000000  00000000  00000159  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000038  00000000  00000000  000001b3  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  000001eb  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000061  00000000  00000000  0000020b  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    00000090  00000000  00000000  0000026c  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000002fc  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  0000030e  2**0
                  CONTENTS, READONLY
 11 .debug_frame  0000002c  00000000  00000000  00000344  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

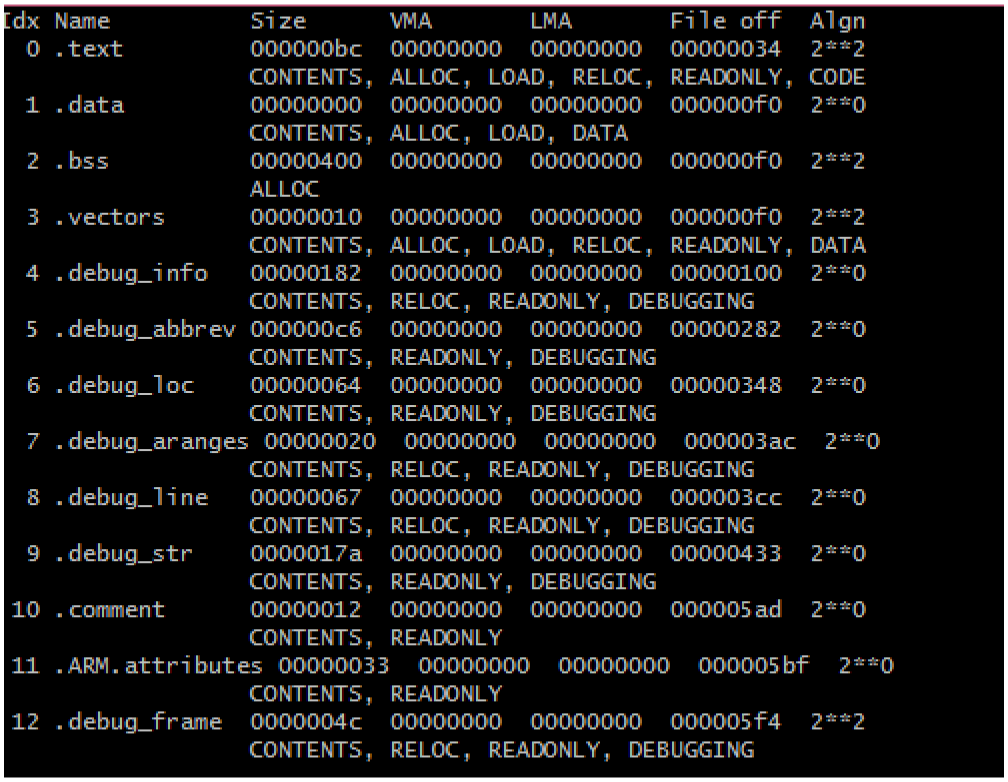
1-text section: size of instruction code =0xc0.

2-data section: size of initialized global array = 0x0 .

3-bss section: size of uninitialized global =0x0.

4-debug sections and other sections.

6.2: startup.o sections headers



```
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000bc  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000f0  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000400  00000000  00000000  000000f0  2**2
                  ALLOC
  3 .vectors      00000010  00000000  00000000  000000f0  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
  4 .debug_info   00000182  00000000  00000000  00000100  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  5 .debug_abbrev 000000c6  00000000  00000000  00000282  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_loc    00000064  00000000  00000000  00000348  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_aranges 00000020  00000000  00000000  000003ac  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_line   00000067  00000000  00000000  000003cc  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  9 .debug_str    0000017a  00000000  00000000  00000433  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .comment      00000012  00000000  00000000  000005ad  2**0
                  CONTENTS, READONLY
 11 .ARM.attributes 00000033  00000000  00000000  000005bf  2**0
                  CONTENTS, READONLY
 12 .debug_frame  0000004c  00000000  00000000  000005f4  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0xbc.

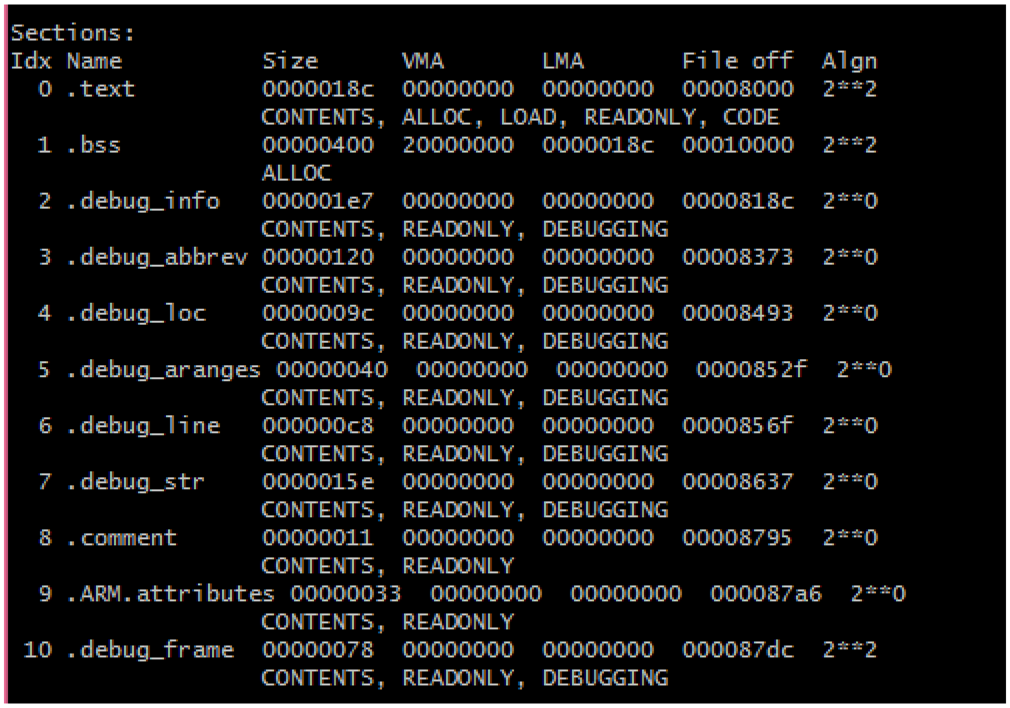
2-data section: size of initialized global array = 0x0 .

3-bss section: size of uninitialized global =0x400.

4-vectors section : size of constant data =0x10.

5-debug sections and other sections.

6.3: elf image sections headers



```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000018c  00000000  00000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .bss          00000400  20000000  0000018c  00010000  2**2
                  ALLOC
  2 .debug_info   000001e7  00000000  00000000  0000818c  2**0
                  CONTENTS, READONLY, DEBUGGING
  3 .debug_abbrev 00000120  00000000  00000000  00008373  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_loc    0000009c  00000000  00000000  00008493  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_aranges 00000040  00000000  00000000  0000852f  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_line   000000c8  00000000  00000000  0000856f  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_str    0000015e  00000000  00000000  00008637  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .comment      00000011  00000000  00000000  00008795  2**0
                  CONTENTS, READONLY
  9 .ARM.attributes 00000033  00000000  00000000  000087a6  2**0
                  CONTENTS, READONLY
 10 .debug_frame  00000078  00000000  00000000  000087dc  2**2
                  CONTENTS, READONLY, DEBUGGING
```

1-text section: size of instruction code =0x18c.

2-bss section: size of initialized global array = 0x400.

3- debug sections and other section.