makefile

```
PROJECT_NAME=Lab_2_ARM_Cortex_M3
CC=arm-none-eabi-
CFLAGS=-mcpu=cortex-m3
INCS=-I .
LIBS=
C_OBJ_SRC=$(wildcard *.c)
C_OBJ=$(C_OBJ_SRC:.c=.o)
S_OBJ_SRC=$(wildcard *.s)
S_OBJ=$(S_OBJ_SRC:.s=.o)

build: $(PROJECT_NAME).bin
    @echo "===============Successful Build==============="

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

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

$(PROJECT_NAME).elf: $(C_OBJ) $(S_OBJ)
    $(CC)ld.exe --script linker_script.ld $(LIBS) $(C_OBJ) $(S_OBJ) -o $(PROJECT_NAME)

$(PROJECT_NAME).bin: $(PROJECT_NAME).elf
    $(CC)objcopy.exe -O binary $< $@

assembly:
    $(CC)objdump.exe -D $(PROJECT_NAME).elf > $(PROJECT_NAME).s
    @echo "===============Generated Assembly from .elf file==============="

clean_all:
    rm *.o *.elf *.bin *.map $(PROJECT_NAME).s
clean_exe:
    rm *.elf *.bin *.map

```

main.c

```
#include <stdint.h>

#define RCC_BASE        0x40021000
#define PORTA_BASE      0x40010800

#define RCC_APB2ENR     *(volatile uint32_t*)(RCC_BASE+0x18)
#define GPIOA_CRH       *(volatile uint32_t*)(PORTA_BASE+0x04)
#define GPIOA_ODR       *(volatile uint32_t*)(PORTA_BASE+0x0C)

int main(void)
{
    RCC_APB2ENR |=1<<2;
    GPIOA_CRH &=0xFF0FFFFF;
    GPIOA_CRH |=0x00200000;
    while (1)
    {
        GPIOA_ODR |=1<<13;
        for (int i=0; i<5000; i++);
        GPIOA_ODR &=~(1<<13);
        for (int i=0; i<5000; i++);
    }
    return 0;
}

```

Part1: startup.s

```
1   .section .vectors
2   .word    0x20001000
3   .word    _reset
4   .word    vector_handler          /* 2 NMI */
5   .word    vector_handler          /* 3 Hard Fault */
6   .word    vector_handler          /* 4 MM Fault */
7   .word    vector_handler          /* 5 Bus Fault */
8   .word    vector_handler          /* 6 Usage Fault */
9            /* To IRQ67 */          /* 7 ..... */
10
11  .section .text
12  _reset:
13     bl main
14     b .
15
16  .thumb_func
17  vector_handler:
18     b _reset
```

linker_script.ld

```
1   ENTRY(.vectors)
2
3   MEMORY
4   {
5       FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
6       SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
7   }
8
9   SECTIONS
10  {
11      .text : {
12              *(.vectors*) /*Expected to be 7*4=28B=1C*/
13              *(.text*) /*Depend on code size in main.c and startup.s*/
14              *.rodata /*Expected to be 0B*/
15      } >FLASH
16
17      .data : {
18              *(.data) /*Expected to be 0B*/
19          } >FLASH
20
21      .bss : {
22              *(.bss) /*Expected to be 0B*/
23          } >SRAM
24      }
```

map_file.map

```
Memory Configuration

Name             Origin             Length
FLASH            0x08000000         0x00020000
SRAM             0x20000000         0x00005000
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000       0xa0
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
 *(.text*)
 .text          0x0800001c       0x7c main.o
                0x0800001c                main
 .text          0x08000098        0x8 startup.o
 *.rodata()

.data           0x080000a0        0x0
 *(.data)
 .data          0x080000a0        0x0 main.o
 .data          0x080000a0        0x0 startup.o

.bss            0x20000000        0x0
 *(.bss)
 .bss           0x20000000        0x0 main.o
 .bss           0x20000000        0x0 startup.o
```

=28B

Disassembly of .elf file

```
Disassembly of section .text:

08000000 <main-0x1c>:
 8000000:   20001000    andcs r1, r0, r0
 8000004:   08000098    stmdaeq  r0, {r3, r4, r7}
 8000008:   0800009f    stmdaeq  r0, {r0, r1, r2, r3, r4, r7}
 800000c:   0800009f    stmdaeq  r0, {r0, r1, r2, r3, r4, r7}
 8000010:   0800009f    stmdaeq  r0, {r0, r1, r2, r3, r4, r7}
 8000014:   0800009f    stmdaeq  r0, {r0, r1, r2, r3, r4, r7}
 8000018:   0800009f    stmdaeq  r0, {r0, r1, r2, r3, r4, r7}
```

- 0x20001000 stored @0x08000000 in FLASH, Initialize SP @0x20001000.
- 0x080000098 stored @0x08000004 in FLASH, 0x080000098 is _reset handler address.
- From 0x080000008 to 0x080000018 which is 28B is our vector table, 0x0800009f is stored in all of them, 0x0800009f is vector_handler address, now we know that vector_handler is alias for 5 handlers.
- Now expected that at runtime CPU go to entry point at 0x08000000 and initialize SP then go to 0x08000004 which branch execution to _reset @0x08000098 and _reset will branch execution to main @0x0800001c.
- Now expected that main instructions start @0x0800001c and ends @0x08000094 then_reset starts @0x08000098 and ends @0x0800009c then vector_handler starts.

- The question here is the address of vector_handler is 0x0800009e and the vector section stores 0x0800009f which is after one byte of 0x0800009e, and the instruction itself e7fb takes 2 bytes which supposed be 0x0800009fe and 0x0800009f
- I tried to debug using QEMU, but the debugger cannot show what happens since it don't stop at breakpoints unless complete the interrupt handler

```
0800001c <main>:
 800001c:   b480        push  {r7}
 800001e:   b083        sub   sp, #12
 8000020:   af00        add   r7, sp, #0
 8000022:   4b1a        ldr   r3, [pc, #104] @ (800008c <main+0x70>)
 8000024:   681b        ldr   r3, [r3, #0]
 8000026:   4a19        ldr   r2, [pc, #100] @ (800008c <main+0x70>)
 8000028:   f043 0304   orr.w r3, r3, #4
 800002c:   6013        str   r3, [r2, #0]
 800002e:   4b18        ldr   r3, [pc, #96]  @ (8000090 <main+0x74>)
 8000030:   681b        ldr   r3, [r3, #0]
 8000032:   4a17        ldr   r2, [pc, #92]  @ (8000090 <main+0x74>)
 8000034:   f423 0370   bic.w r3, r3, #15728640 @ 0xf00000
 8000038:   6013        str   r3, [r2, #0]
 800003a:   4b15        ldr   r3, [pc, #84]  @ (8000090 <main+0x74>)
 800003c:   681b        ldr   r3, [r3, #0]
 800003e:   4a14        ldr   r2, [pc, #80]  @ (8000090 <main+0x74>)
 8000040:   f443 1300   orr.w r3, r3, #2097152  @ 0x200000
 8000044:   6013        str   r3, [r2, #0]
 8000046:   4b13        ldr   r3, [pc, #76]  @ (8000094 <main+0x78>)
 8000048:   681b        ldr   r3, [r3, #0]
 800004a:   4a12        ldr   r2, [pc, #72]  @ (8000094 <main+0x78>)
 800004c:   f443 5300   orr.w r3, r3, #8192  @ 0x2000
 8000050:   6013        str   r3, [r2, #0]
 8000052:   2300        movs  r3, #0
 8000054:   607b        str   r3, [r7, #4]
 8000056:   e002        b.n   800005e <main+0x42>
 8000058:   687b        ldr   r3, [r7, #4]
 800005a:   3301        adds  r3, #1
 800005c:   607b        str   r3, [r7, #4]
 800005e:   687b        ldr   r3, [r7, #4]
 8000060:   f241 3287   movw  r2, #4999   @ 0x1387
 8000064:   4293        cmp   r3, r2
 8000066:   ddf7        ble.n 8000058 <main+0x3c>
 8000068:   4b0a        ldr   r3, [pc, #40]  @ (8000094 <main+0x78>)
 800006a:   681b        ldr   r3, [r3, #0]

 800006c:   4a09        ldr   r2, [pc, #36]  @ (8000094 <main+0x78>)
 800006e:   f423 5300   bic.w r3, r3, #8192  @ 0x2000
 8000072:   6013        str   r3, [r2, #0]
 8000074:   2300        movs  r3, #0
 8000076:   603b        str   r3, [r7, #0]
 8000078:   e002        b.n   8000080 <main+0x64>
 800007a:   683b        ldr   r3, [r7, #0]
 800007c:   3301        adds  r3, #1
 800007e:   603b        str   r3, [r7, #0]
 8000080:   683b        ldr   r3, [r7, #0]
 8000082:   f241 3287   movw  r2, #4999   @ 0x1387
 8000086:   4293        cmp   r3, r2
 8000088:   ddf7        ble.n 800007a <main+0x5e>
 800008a:   e7dc        b.n   8000046 <main+0x2a>
 800008c:   40021018    andmi r1, r2, r8, lsl r0
 8000090:   40010804    andmi r0, r1, r4, lsl #16
 8000094:   4001080c    andmi r0, r1, ip, lsl #16

08000098 <_reset>:
 8000098:   f7ff ffc0   bl 800001c <main>
 800009c:   e7fe        b.n   800009c <_reset+0x4>

0800009e <vector_handler>:
 800009e:   e7fb        b.n   8000098 <_reset>
```

Part2: startup.c

Without alias:

```c
#include <stdint.h>
#define stack_top 0x20001000

extern int main(void);
//void Default_handler(void);

void Reset_handler(void);
void NMI_handler(void)__attribute__((weak));
void HW_fault_handler(void)__attribute__((weak));
void MM_fault_handler(void)__attribute__((weak));
void Bus_fault_handler(void)__attribute__((weak));
void Usage_fault_handler(void)__attribute__((weak));

uint32_t vectors[]__attribute__((section(".vectors")))={
    stack_top,
    (uint32_t) &Reset_handler,
    (uint32_t) &NMI_handler,
    (uint32_t) &HW_fault_handler,
    (uint32_t) &MM_fault_handler,
    (uint32_t) &Bus_fault_handler,
    (uint32_t) &Usage_fault_handler,
};

void Reset_handler(void){
    main();
}
void NMI_handler(void){Reset_handler();}
void HW_fault_handler(void){Reset_handler();}
void MM_fault_handler(void){Reset_handler();}
void Bus_fault_handler(void){Reset_handler();}
void Usage_fault_handler(void){Reset_handler();}
/*
void Default_handler(void){
    Reset_handler();
}
*/
```

linker_script.ld

```
ENTRY(.vectors)

MEMORY
{
    FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
    .text : {
            *(.vectors*) /*Expected to be 7*4=28B=1C*/
            *(.text*) /*Depend on code size in main.c and startup.c*/
            *.rodata /*Expected to be 0B*/
    } >FLASH

    .data : {
            *(.data) /*Expected to be 0B*/
        } >FLASH

    .bss : {
            *(.bss) /*Expected to be 0B*/
        } >SRAM
    }
```

map_file.map

```
1
2   Memory Configuration
3
4   Name             Origin             Length             Attributes
5   FLASH            0x08000000         0x00020000         xr
6   SRAM             0x20000000         0x00005000         xrw
7   *default*        0x00000000         0xffffffff
8
9   Linker script and memory map
10
11
12  .text           0x08000000       0xe0
13   *(.vectors*)
14   .vectors        0x08000000       0x1c startup.o
15                   0x08000000                vectors
16   *(.text*)
17   .text           0x0800001c       0x7c main.o
18                   0x0800001c                main
19   .text           0x08000098       0x48 startup.o
20                   0x08000098                Reset_handler
21                   0x080000a4                NMI_handler
22                   0x080000b0                HW_fault_handler
23                   0x080000bc                MM_fault_handler
24                   0x080000c8                Bus_fault_handler
25                   0x080000d4                Usage_fault_handler
26   *.rodata()
```

```
46   .data           0x080000e0        0x0
47    *(.data)
48    .data          0x080000e0        0x0 main.o
49    .data          0x080000e0        0x0 startup.o

54   .bss            0x20000000        0x0
55    *(.bss)
56    .bss           0x20000000        0x0 main.o
57    .bss           0x20000000        0x0 startup.o
```

- Each fault handler has its own address.

Disassembly of .elf file

```
5   Disassembly of section .text:
6
7   08000000 <vectors>:
8    8000000:   20001000    andcs r1, r0, r0
9    8000004:   08000099    stmdaeq  r0, {r0, r3, r4, r7}
10   8000008:   080000a5    stmdaeq  r0, {r0, r2, r5, r7}
11   800000c:   080000b1    stmdaeq  r0, {r0, r4, r5, r7}
12   8000010:   080000bd    stmdaeq  r0, {r0, r2, r3, r4, r5, r7}
13   8000014:   080000c9    stmdaeq  r0, {r0, r3, r6, r7}
14   8000018:   080000d5    stmdaeq  r0, {r0, r2, r4, r6, r7}
15
```

```
0800001c <main>:
 800001c:   b480        push  {r7}
 800001e:   b083        sub   sp, #12
 8000020:   af00        add   r7, sp, #0
 8000022:   4b1a        ldr   r3, [pc, #104] @ (800008c <main+0x70>)
 8000024:   681b        ldr   r3, [r3, #0]
 8000026:   4a19        ldr   r2, [pc, #100] @ (800008c <main+0x70>)
 8000028:   f043 0304   orr.w r3, r3, #4
 800002c:   6013        str   r3, [r2, #0]
 800002e:   4b18        ldr   r3, [pc, #96]  @ (8000090 <main+0x74>)
 8000030:   681b        ldr   r3, [r3, #0]
 8000032:   4a17        ldr   r2, [pc, #92]  @ (8000090 <main+0x74>)
 8000034:   f423 0370   bic.w r3, r3, #15728640 @ 0xf00000
 8000038:   6013        str   r3, [r2, #0]
 800003a:   4b15        ldr   r3, [pc, #84]  @ (8000090 <main+0x74>)
 800003c:   681b        ldr   r3, [r3, #0]
 800003e:   4a14        ldr   r2, [pc, #80]  @ (8000090 <main+0x74>)
 8000040:   f443 1300   orr.w r3, r3, #2097152  @ 0x200000
 8000044:   6013        str   r3, [r2, #0]
 8000046:   4b13        ldr   r3, [pc, #76]  @ (8000094 <main+0x78>)
 8000048:   681b        ldr   r3, [r3, #0]
 800004a:   4a12        ldr   r2, [pc, #72]  @ (8000094 <main+0x78>)
 800004c:   f443 5300   orr.w r3, r3, #8192  @ 0x2000
 8000050:   6013        str   r3, [r2, #0]
 8000052:   2300        movs  r3, #0
 8000054:   607b        str   r3, [r7, #4]
 8000056:   e002        b.n   800005e <main+0x42>
 8000058:   687b        ldr   r3, [r7, #4]
 800005a:   3301        adds  r3, #1
 800005c:   607b        str   r3, [r7, #4]
 800005e:   687b        ldr   r3, [r7, #4]
 8000060:   f241 3287   movw  r2, #4999   @ 0x1387
 8000064:   4293        cmp   r3, r2
 8000066:   ddf7        ble.n 8000058 <main+0x3c>
 8000068:   4b0a        ldr   r3, [pc, #40]  @ (8000094 <main+0x78>)
 800006a:   681b        ldr   r3, [r3, #0]
 800006c:   4a09        ldr   r2, [pc, #36]  @ (8000094 <main+0x78>)
 800006e:   f423 5300   bic.w r3, r3, #8192  @ 0x2000
 8000072:   6013        str   r3, [r2, #0]
 8000074:   2300        movs  r3, #0
 8000076:   603b        str   r3, [r7, #0]
 8000078:   e002        b.n   8000080 <main+0x64>
 800007a:   683b        ldr   r3, [r7, #0]
 800007c:   3301        adds  r3, #1
 800007e:   603b        str   r3, [r7, #0]
 8000080:   683b        ldr   r3, [r7, #0]
 8000082:   f241 3287   movw  r2, #4999   @ 0x1387
 8000086:   4293        cmp   r3, r2
 8000088:   ddf7        ble.n 800007a <main+0x5e>
 800008a:   e7dc        b.n   8000046 <main+0x2a>
 800008c:   40021018    andmi r1, r2, r8, lsl r0
 8000090:   40010804    andmi r0, r1, r4, lsl #16
 8000094:   4001080c    andmi r0, r1, ip, lsl #16


 08000098 <Reset_handler>:
  8000098:   b580        push  {r7, lr}
  800009a:   af00        add   r7, sp, #0
  800009c:   f7ff ffbe   bl 800001c <main>
  80000a0:   bf00        nop
  80000a2:   bd80        pop   {r7, pc}

 080000a4 <NMI_handler>:
  80000a4:   b580        push  {r7, lr}
  80000a6:   af00        add   r7, sp, #0
  80000a8:   f7ff fff6   bl 8000098 <Reset_handler>
  80000ac:   bf00        nop
  80000ae:   bd80        pop   {r7, pc}

 080000b0 <HW_fault_handler>:
  80000b0:   b580        push  {r7, lr}
  80000b2:   af00        add   r7, sp, #0
  80000b4:   f7ff fff0   bl 8000098 <Reset_handler>
  80000b8:   bf00        nop
  80000ba:   bd80        pop   {r7, pc}

 080000bc <MM_fault_handler>:
  80000bc:   b580        push  {r7, lr}
  80000be:   af00        add   r7, sp, #0
  80000c0:   f7ff ffea   bl 8000098 <Reset_handler>
  80000c4:   bf00        nop
  80000c6:   bd80        pop   {r7, pc}

 080000c8 <Bus_fault_handler>:
  80000c8:   b580        push  {r7, lr}
  80000ca:   af00        add   r7, sp, #0
  80000cc:   f7ff ffe4   bl 8000098 <Reset_handler>
  80000d0:   bf00        nop
  80000d2:   bd80        pop   {r7, pc}

 080000d4 <Usage_fault_handler>:
  80000d4:   b580        push  {r7, lr}
  80000d6:   af00        add   r7, sp, #0
  80000d8:   f7ff ffde   bl 8000098 <Reset_handler>
  80000dc:   bf00        nop
  80000de:   bd80        pop   {r7, pc}
```

With alias:

```c
#include <stdint.h>
#define stack_top 0x20001000

extern int main(void);
void Default_handler(void);

void Reset_handler(void);
void NMI_handler(void)__attribute__((weak,alias("Default_handler")));
void HW_fault_handler(void)__attribute__((weak,alias("Default_handler")));
void MM_fault_handler(void)__attribute__((weak,alias("Default_handler")));
void Bus_fault_handler(void)__attribute__((weak,alias("Default_handler")));
void Usage_fault_handler(void)__attribute__((weak,alias("Default_handler")));

uint32_t vectors[]__attribute__((section(".vectors")))={
    stack_top,
    (uint32_t) &Reset_handler,
    (uint32_t) &NMI_handler,
    (uint32_t) &HW_fault_handler,
    (uint32_t) &MM_fault_handler,
    (uint32_t) &Bus_fault_handler,
    (uint32_t) &Usage_fault_handler,
};

void Reset_handler(void){
    main();
}

void Default_handler(void){
    Reset_handler();
}
```

linker_script.ld

```
ENTRY(.vectors)

MEMORY
{
    FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
    .text : {
            *(.vectors*) /*Expected to be 7*4=28B=1C*/
            *(.text*) /*Depend on code size in main.c and startup.c*/
            *.rodata /*Expected to be 0B*/
    } >FLASH

    .data : {
            *(.data) /*Expected to be 0B*/
        } >FLASH

    .bss : {
            *(.bss) /*Expected to be 0B*/
        } >SRAM
    }
```

map_file.map

```
Memory Configuration

Name             Origin             Length             Attributes
FLASH            0x08000000         0x00020000         xr
SRAM             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000       0xb0
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
                0x08000000                vectors
 *(.text*)
 .text          0x0800001c       0x7c main.o
                0x0800001c                main
 .text          0x08000098       0x18 startup.o
                0x08000098                Reset_handler
                0x080000a4                Bus_fault_handler
                0x080000a4                HW_fault_handler
                0x080000a4                Usage_fault_handler
                0x080000a4                MM_fault_handler
                0x080000a4                Default_handler
                0x080000a4                NMI_handler
 *.rodata()

 .data          0x080000b0        0x0
  *(.data)
  .data         0x080000b0        0x0 main.o
  .data         0x080000b0        0x0 startup.o

 .bss           0x20000000        0x0
  *(.bss)
  .bss          0x20000000        0x0 main.o
  .bss          0x20000000        0x0 startup.o
```

- All of fault handlers have the same address of Default_handler, since all of them are alias for Default_handler

Disassembly of .elf file

```
Disassembly of section .text:

08000000 <vectors>:
 8000000:   20001000    andcs r1, r0, r0
 8000004:   08000099    stmdaeq  r0, {r0, r3, r4, r7}
 8000008:   080000a5    stmdaeq  r0, {r0, r2, r5, r7}
 800000c:   080000a5    stmdaeq  r0, {r0, r2, r5, r7}
 8000010:   080000a5    stmdaeq  r0, {r0, r2, r5, r7}
 8000014:   080000a5    stmdaeq  r0, {r0, r2, r5, r7}
 8000018:   080000a5    stmdaeq  r0, {r0, r2, r5, r7}

```

```
0800001c <main>:
 800001c:   b480        push  {r7}
 800001e:   b083        sub   sp, #12
 8000020:   af00        add   r7, sp, #0
 8000022:   4b1a        ldr   r3, [pc, #104] @ (800008c <main+0x70>)
 8000024:   681b        ldr   r3, [r3, #0]
 8000026:   4a19        ldr   r2, [pc, #100] @ (800008c <main+0x70>)
 8000028:   f043 0304   orr.w r3, r3, #4
 800002c:   6013        str   r3, [r2, #0]
 800002e:   4b18        ldr   r3, [pc, #96]  @ (8000090 <main+0x74>)
 8000030:   681b        ldr   r3, [r3, #0]
 8000032:   4a17        ldr   r2, [pc, #92]  @ (8000090 <main+0x74>)
 8000034:   f423 0370   bic.w r3, r3, #15728640 @ 0xf00000
 8000038:   6013        str   r3, [r2, #0]
 800003a:   4b15        ldr   r3, [pc, #84]  @ (8000090 <main+0x74>)
 800003c:   681b        ldr   r3, [r3, #0]
 800003e:   4a14        ldr   r2, [pc, #80]  @ (8000090 <main+0x74>)
 8000040:   f443 1300   orr.w r3, r3, #2097152  @ 0x200000
 8000044:   6013        str   r3, [r2, #0]
 8000046:   4b13        ldr   r3, [pc, #76]  @ (8000094 <main+0x78>)
 8000048:   681b        ldr   r3, [r3, #0]
 800004a:   4a12        ldr   r2, [pc, #72]  @ (8000094 <main+0x78>)
 800004c:   f443 5300   orr.w r3, r3, #8192  @ 0x2000
 8000050:   6013        str   r3, [r2, #0]
 8000052:   2300        movs  r3, #0
 8000054:   607b        str   r3, [r7, #4]
 8000056:   e002        b.n   800005e <main+0x42>
 8000058:   687b        ldr   r3, [r7, #4]
 800005a:   3301        adds  r3, #1
 800005c:   607b        str   r3, [r7, #4]
 800005e:   687b        ldr   r3, [r7, #4]
 8000060:   f241 3287   movw  r2, #4999   @ 0x1387
 8000064:   4293        cmp   r3, r2
 8000066:   ddf7        ble.n 8000058 <main+0x3c>
 8000068:   4b0a        ldr   r3, [pc, #40]  @ (8000094 <main+0x78>)
 800006a:   681b        ldr   r3, [r3, #0]
 800006c:   4a09        ldr   r2, [pc, #36]  @ (8000094 <main+0x78>)
 800006e:   f423 5300   bic.w r3, r3, #8192  @ 0x2000
 8000072:   6013        str   r3, [r2, #0]
 8000074:   2300        movs  r3, #0
 8000076:   603b        str   r3, [r7, #0]
 8000078:   e002        b.n   8000080 <main+0x64>
 800007a:   683b        ldr   r3, [r7, #0]
 800007c:   3301        adds  r3, #1
 800007e:   603b        str   r3, [r7, #0]
 8000080:   683b        ldr   r3, [r7, #0]
 8000082:   f241 3287   movw  r2, #4999   @ 0x1387
 8000086:   4293        cmp   r3, r2
 8000088:   ddf7        ble.n 800007a <main+0x5e>
 800008a:   e7dc        b.n   8000046 <main+0x2a>
 800008c:   40021018    andmi r1, r2, r8, lsl r0
 8000090:   40010804    andmi r0, r1, r4, lsl #16
 8000094:   4001080c    andmi r0, r1, ip, lsl #16

 08000098 <Reset_handler>:
  8000098:   b580        push  {r7, lr}
  800009a:   af00        add   r7, sp, #0
  800009c:   f7ff ffbe   bl 800001c <main>
  80000a0:   bf00        nop
  80000a2:   bd80        pop   {r7, pc}

 080000a4 <Default_handler>:
  80000a4:   b580        push  {r7, lr}
  80000a6:   af00        add   r7, sp, #0
  80000a8:   f7ff fff6   bl 8000098 <Reset_handler>
  80000ac:   bf00        nop
  80000ae:   bd80        pop   {r7, pc}
```

Part3: startup.c (alias & .data & .bss)

main.c

```c
#include <stdint.h>

#define RCC_BASE        0x40021000
#define PORTA_BASE      0x40010800

#define RCC_APB2ENR     *(volatile uint32_t*)(RCC_BASE+0x18)
#define GPIOA_CRH       *(volatile uint32_t*)(PORTA_BASE+0x04)
#define GPIOA_ODR       *(volatile uint32_t*)(PORTA_BASE+0x0C)

volatile char DataArr[]={1,2,3}; //for 3B data section
volatile struct A {
    char ch;
    uint32_t var;
} BssStruct; //for 8B bss section


int main(void)
{
    RCC_APB2ENR |=1<<2;
    GPIOA_CRH &=0xFF0FFFFF;
    GPIOA_CRH |=0x00200000;
    while (1)
    {
        GPIOA_ODR |=1<<13;
        for (int i=0; i<5000; i++);
        GPIOA_ODR &=~(1<<13);
        for (int i=0; i<5000; i++);
    }
    return 0;
}
```

startup.c

```c
#include <stdint.h>
#define NULL (void*)0

extern int main(void);
void Default_handler(void);

extern uint32_t stack_top;
void Reset_handler(void);
void NMI_handler(void)__attribute__((weak,alias("Default_handler")));
void HW_fault_handler(void)__attribute__((weak,alias("Default_handler")));
void MM_fault_handler(void)__attribute__((weak,alias("Default_handler")));
void Bus_fault_handler(void)__attribute__((weak,alias("Default_handler")));
void Usage_fault_handler(void)__attribute__((weak,alias("Default_handler")));

uint32_t vectors[]__attribute__((section(".vectors")))={
    (uint32_t) &stack_top,
    (uint32_t) &Reset_handler,
    (uint32_t) &NMI_handler,
    (uint32_t) &HW_fault_handler,
    (uint32_t) &MM_fault_handler,
    (uint32_t) &Bus_fault_handler,
    (uint32_t) &Usage_fault_handler,
};

extern uint32_t _END_text;
extern uint32_t _START_data;
extern uint32_t _END_data;
extern uint32_t _START_bss;
extern uint32_t _END_bss;

```

```c
void Reset_handler(void){

    uint32_t section_size;
    uint8_t *p_src, *p_dist;

    //copy .data to SRAM
    section_size=(uint8_t*)&_END_data - (uint8_t*)&_START_data;
    p_src=(uint8_t*)&_END_text;
    p_dist=(uint8_t*)&_START_data;
    for (int i = 0; i < section_size; i++, p_dist++, p_src++)
    {
        *p_dist=*p_src;
    }

    //create .bss in SRAM with 0
    section_size=(uint8_t*)&_END_bss - (uint8_t*)&_START_bss;
    p_src=NULL;
    p_dist=(uint8_t*)&_START_bss;
    for (int i = 0; i < section_size; i++, p_dist++, p_src++)
    {
        *p_dist=0;
    }

    //branch to main
    main();
}

void Default_handler(void){
    Reset_handler();
}
```

linker_script.ld

```
ENTRY(vectors)

MEMORY
{
    FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
    .text : {
            *(.vectors*) /*Expected to be 7*4=28B=1C*/
            *(.text*) /*Depend on code size in main.c and startup.c*/
            *.rodata /*Expected to be 0B*/
            _END_text = . ;
    } >FLASH


    .data : ALIGN(4) {
            _START_data = . ;
            *(.data) /*Expected to be 0B*/
            . = ALIGN(4);
            _END_data = . ;
        }> SRAM AT>FLASH

    .bss : ALIGN(4) {
            _START_bss = . ;
            *(.bss) /*Expected to be 0B*/
            . = ALIGN(4);
            _END_bss = . ;
        } >SRAM
        . = . + 0x1000; /*not inside bss because this assignment change location counter .*/
        stack_top = . ; /*if inside bss the size of bss will increase 1000B*/
}
```

map_file.map

```
Memory Configuration

Name             Origin             Length             Attributes
FLASH            0x08000000         0x00020000         xr
SRAM             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000      0x138
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
                0x08000000                vectors
 *(.text*)
 .text          0x0800001c       0x7c main.o
                0x0800001c                main
 .text          0x08000098       0xa0 startup.o
                0x08000098                Reset_handler
                0x0800012c                Bus_fault_handler
                0x0800012c                HW_fault_handler
                0x0800012c                Usage_fault_handler
                0x0800012c                MM_fault_handler
                0x0800012c                Default_handler
                0x0800012c                NMI_handler
 *.rodata()
                0x08000138                        _END_text = .

.data           0x20000000        0x4 load address 0x08000138
                0x20000000                        _START_data = .
 *(.data)
 .data          0x20000000        0x3 main.o
                0x20000000             DataArr
 .data          0x20000003        0x0 startup.o
                0x20000004                        . = ALIGN (0x4)
 *fill*         0x20000003        0x1
                0x20000004                        _END_data = .
```

- *fill* of 1B is for Alignment, the size of global initialized variable is 3B.

```
.bss            0x20000004        0x8 load address 0x0800013c
                0x20000004                        _START_bss = .
 *(.bss)
 .bss           0x20000004        0x8 main.o
                0x20000004             BssStruct
 .bss           0x2000000c        0x0 startup.o
                0x2000000c                        . = ALIGN (0x4)
                0x2000000c                        _END_bss = .
                0x2000100c                        . = (. + 0x1000)
                0x2000100c                        stack_top = .
```

- The size of initialized struct taken by the 8 because of auto alignment in C structures.

sections of .elf file

```
$ arm-none-eabi-objdump -h Lab_2_ARM_Cortex_M3.elf

Lab_2_ARM_Cortex_M3.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000138  08000000  08000000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  08000138  00002000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000008  20000004  0800013c  00002004  2**2
                  ALLOC
  3 .comment      00000043  00000000  00000000  00002004  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 0000002d  00000000  00000000  00002047  2**0
                  CONTENTS, READONLY
```

Symbols of .elf file

```
$ arm-none-eabi-nm.exe Lab_2_ARM_Cortex_M3.elf
2000000c B _END_bss
20000004 D _END_data
08000138 T _END_text
20000004 B _START_bss
20000000 D _START_data
20000004 B BssStruct
0800012c W Bus_fault_handler
20000000 D DataArr
0800012c T Default_handler
0800012c W HW_fault_handler
0800001c T main
0800012c W MM_fault_handler
0800012c W NMI_handler
08000098 T Reset_handler
2000100c B stack_top
0800012c W Usage_fault_handler
08000000 T vectors
```