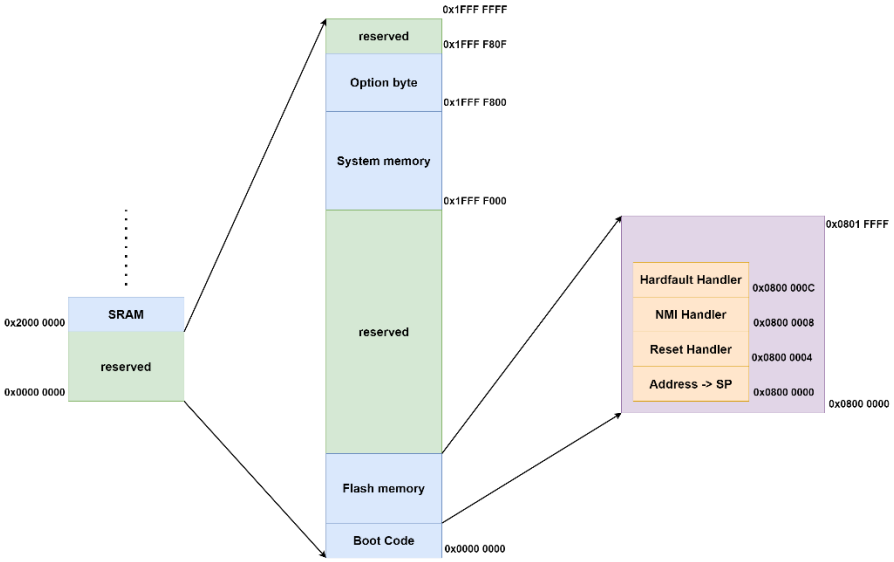
Lab2 – Bare-metal toggle led on STM32:

This lab covers creating a bare-metal Software to blink a LED in ARM STM32 MCU (Cortex M3).

1. Define **Interrupt vectors** Section
2. Copy Data from **ROM** → **RAM**
3. Initialize **Data Area**
4. Initialize **Stack**
5. Create a **reset section** and call main( ).

We need first to determine where the flash address we will place the startup code at:



Makefile:

```
#@arm-none-eabi-#@Copyright: Ahmed Hassan

CC=arm-none-eabi-
CFLAGS=-mcpu=cortex-m3 -gdwarf-2 #included debugger for proteus
INCS=-I .
LIBS=
SRC=$(wildcard *.c)
OBJ=$(SRC:.c=.o)
As=$(wildcard *.s)
AsOBJ=$(As:.s=.o)
Project_name=Learn_in_depth_cortex_m3

all: $(Project_name).bin
    @echo "<<============== Build Complete ==============>>"

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

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

$(Project_name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) -Map=output.map $(OBJ) $(AsOBJ) -o $@

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

clean_all:
    rm *.o *.elf *.bin

clean:
    rm *.elf *.bin
```

Startup Code:

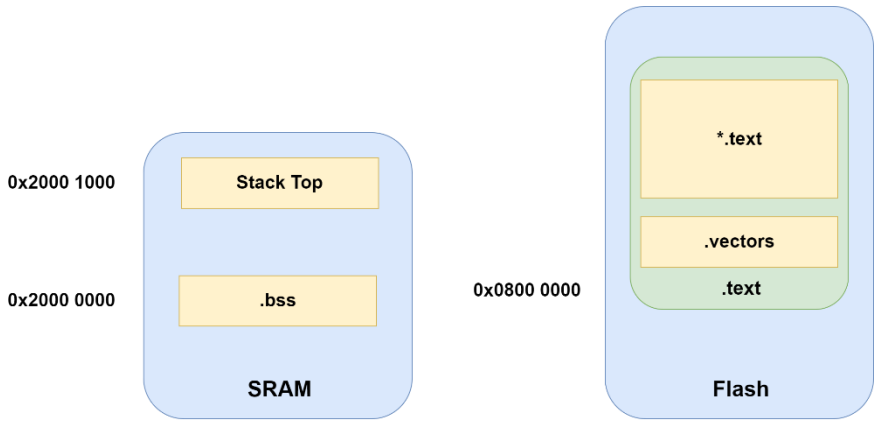
```
/* startup_cortexM3.s
Eng.Ahmed Hassan
*/

/* ===============SRAM 0x2000 0000=============== */
.section .vectors       /* Assembler command to set object section */

.word 0x20001000        /* For Stack */
.word _reset            /* For Vector Handler */
.word Vector_handler    /* 2 NMI */
.word Vector_handler    /* 3 Hard Fault */
.word Vector_handler    /* 4 MM Fault */
.word Vector_handler    /* 5 Bus Fault */
.word Vector_handler    /* 6 Usage Fault */
.word Vector_handler    /* 7 RESERVED */
.word Vector_handler    /* 8 RESERVED */
.word Vector_handler    /* 9 RESERVED */
.word Vector_handler    /* 10 RESERVED */
.word Vector_handler    /* 11 SV Call */
.word Vector_handler    /* 12 Debug Resurved */
.word Vector_handler    /* 13 RESERVED */
.word Vector_handler    /* 14 PendSV */
.word Vector_handler    /* 15 SysTick */
.word Vector_handler    /* 16 IRQ0 */
.word Vector_handler    /* 17 IRQ1 */
.word Vector_handler    /* 18 IRQ2 */
.word Vector_handler    /* 19 .... */
                        /* On until IRQ67 */

.section .text

_reset:
    bl main
    b .         /* Branch to yourself */

.thumb_func     /* Enable thumb mode */

Vector_handler:
    b _reset
```

Table 61. Vector table for connectivity line devices

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060

Linker Script:

We want to mimic the following memory segments:



```
/* Linker Script CortexM3
Eng. Ahmed Hassan
*/

MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128k
    SRAM (RWX) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
    .text :
    {
        *(.vectors*)    /* Get .vectors from any object files */
        *(.text*)       /* Get .text from any object files */
        *(.rodata)      /* Get .rodata from any object files */
    }> flash

    .data :
    {
        *(.data)
    }> flash

    .bss :
    {
        *(.bss)
    }> SRAM
}
```

Startup using C code:

As (CortexM3) can initialize the **SP with the first 4 bytes**, so we can write **startup by C code**.

Functional Attribute: weak and alias in embedded c:

```
$ arm-none-eabi-nm.exe Learn_in_depth_cortex_m3.elf
080000c8 T Bus_Fault
080000e0 T const_variables
080000e8 D g_variables
080000b0 T H_fault_Handler
0800001c T main
080000bc T MM_Fault_Handler
080000a4 T NMI_Handler
080000e4 D ODR_A
08000098 T Reset_Handler
080000d4 T Usage_Fault_Handler
08000000 T vectors
```

Managing **many ISRs (Interrupt Service Routines)** manually becomes **tedious**. This is where GCC attributes like **__attribute__((weak))** and **__attribute__((alias("...")))** become extremely helpful.

If you write a handler for every possible exception and interrupt, even if you don't use most of them, you'll end up with lots of **empty** or **duplicate** code just to avoid linker errors.

- **__attribute__((weak)) (Weak Linking):**
  - Allows a function to be overridden by a strong (non-weak) definition elsewhere.
  - Provides a default implementation that can be replaced by the user.

- **__attribute__((alias)) (Function Aliasing)**
  - Creates an alternative name for a function.
  - Useful for backward compatibility or generic function redirection.

Now we can see that **handlers with the pragma commands** have the **same symbol** as the **Default_Handler**:

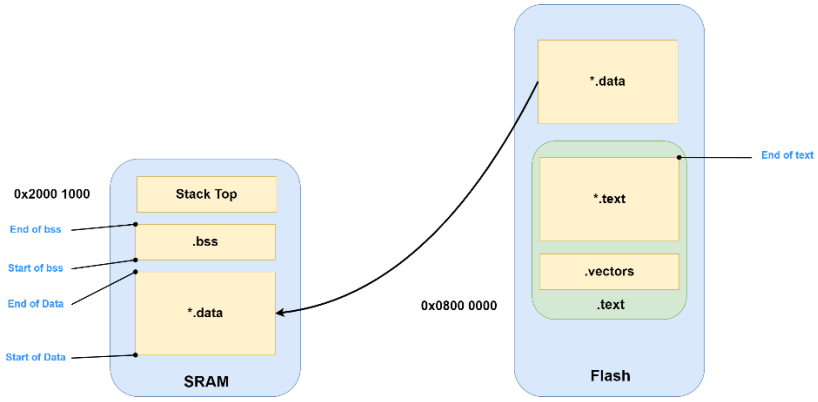
```
$ arm-none-eabi-nm.exe Learn_in_depth_cortex_m3.elf
08000098 W Bus_Fault
080000b0 T const_variables
08000098 T Default_Handler
080000b8 D g_variables
08000098 W H_fault_Handler
0800001c T main
08000098 W MM_Fault_Handler
08000098 W NMI_Handler
080000b4 D ODR_A
080000a4 T Reset_Handler
08000098 W Usage_Fault_Handler
08000000 T vectors
```

Startup.c code:

```c
//Startup.c
//Eng.Ahmed Hassan

#include <stdint.h>

extern int main (void);

void Default_Handler();
void Reset_Handler();
void NMI_Handler() __attribute__ ((weak, alias ("Default_Handler")));
void H_fault_Handler() __attribute__ ((weak, alias ("Default_Handler")));
void MM_Fault_Handler() __attribute__ ((weak, alias ("Default_Handler")));
void Bus_Fault() __attribute__ ((weak, alias ("Default_Handler")));
void Usage_Fault_Handler() __attribute__ ((weak, alias ("Default_Handler")));

//Pragma to create a section with the name ".vectors" so the linker can assign to correct address
uint32_t vectors[] __attribute__ ((section(".vectors"))) = {

(uint32_t)  0x20001000,         //The vector handler start at 0x20001000
(uint32_t)  &Reset_Handler,
(uint32_t)  &NMI_Handler,
(uint32_t)  &H_fault_Handler,
(uint32_t)  &MM_Fault_Handler,
(uint32_t)  &Bus_Fault,
(uint32_t)  &Usage_Fault_Handler

};

void Default_Handler()
{
    Reset_Handler();
}

void Reset_Handler()
{
    main();
}
```

How to copy (data and create .bss sections):

Now we want to copy the .data from flash to SRAM and create .bss in SRAM.



As a start, we can use the same address value for "**End of text**" = "**Start of Data**". We will also use **virtual memory** to calculate the **size** of data and bss sections.

Modified Linker script:

```
/* Linker Script CortexM3
Eng. Ahmed Hassan
*/

MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128k
    SRAM (RWX) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
    .text :
    {
        *(.vectors*)        /* Get .vectors from any object files */
        *(.text*)           /* Get .text from any object files */
        *(.rodata)          /* Get .rodata from any object files */
        _E_text = . ;        /* End of text */
    }> flash                /* Both VM and LM in flash*/

    .data :
    {
        _S_DATA = . ;        /* Start of data right after .text */
        *(.data)
        . = ALIGN(4) ;       /* Enable memory allignment */
        _E_DATA = . ;        /* End of data */
    }> SRAM AT> flash       /* Virtual address at SRAM and at burning start in flash */

    .bss :
    {
        _S_bss = . ;         /* Start of bss */
        *(.bss)
        _E_bss = . ;         /* End of bss */
        . = ALIGN(4) ;
        . = . + 0x1000 ;    /* Make sure to not overwrite the stack top */
        _stack_top = . ;
    }> SRAM                 /* Both VM and LM in SRAM*/
}
```

Modified Startup.c:

```c
//Startup.c
//Eng.Ahmed Hassan

#include <stdint.h>

extern int main (void);

void Default_Handler();
void Reset_Handler();
void NMI_Handler() __attribute__ ((weak, alias ("Default_Handler")));
void H_fault_Handler() __attribute__ ((weak, alias ("Default_Handler")));
void MM_Fault_Handler() __attribute__ ((weak, alias ("Default_Handler")));
void Bus_Fault() __attribute__ ((weak, alias ("Default_Handler")));
void Usage_Fault_Handler() __attribute__ ((weak, alias ("Default_Handler")));

extern uint32_t _stack_top;

//Pragma to create a section with the name ".vectors" so the linker can assign to correct address
uint32_t vectors[] __attribute__ ((section(".vectors"))) = {

(uint32_t)  &_stack_top,         //The vector handler start at 0x20001000
(uint32_t)  &Reset_Handler,
(uint32_t)  &NMI_Handler,
(uint32_t)  &H_fault_Handler,
(uint32_t)  &MM_Fault_Handler,
(uint32_t)  &Bus_Fault,
(uint32_t)  &Usage_Fault_Handler

};

//Data and bss sections from the linker script
extern uint32_t _E_text;
extern uint32_t _S_DATA;
extern uint32_t _E_DATA;
extern uint32_t _S_bss;
extern uint32_t _E_bss;

void Reset_Handler()
{
    //These are not variables but symbols so we act as if they are addresses
    //In case data is not alligned, we pass byte by byte while casting
    uint32_t DATA_size = (uint8_t*)&_E_DATA - (uint8_t*)&_S_DATA;
    uint8_t* P_src = (uint8_t*)&_E_text;
    uint8_t* P_dst = (uint8_t*)&_S_DATA;

    //Copy data section from flash to SRAM
    for (int i = 0; i < DATA_size; i++)
    {
        *((uint8_t*)P_dst++) = *((uint8_t*)P_src++);
    }

    //init .bss section in SRAM = 0
    uint32_t bss_size = (uint8_t*)&_E_bss - (uint8_t*)&_S_bss;
    P_dst = (uint8_t*)&_S_bss;

    for (int i = 0; i < bss_size; i++)
    {
        *((uint8_t*)P_dst++) = (uint8_t)0;
    }

    //Jump to main()
    main();
}

void Default_Handler()
{
    Reset_Handler();
}
```