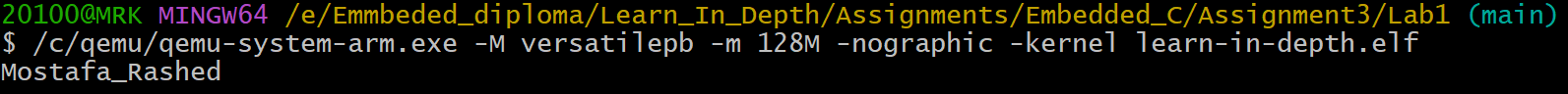
Report

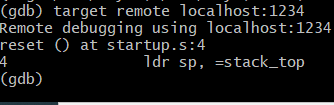
Lab1(gdb&Makefile) Lab2(Startup.s&Startup.c)

Lab1(gdb&Makefile) :

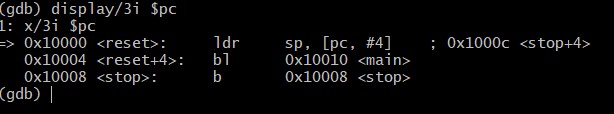
* First we will open gdb circuit in qemo tool for board that we debug on called versatilepb using this command :



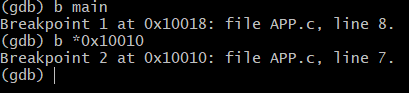
* + As we know to connect to gdb server on board you must have IP address and port number
  + In our case we use qemu tool to virtually debug our code so the IP address will be our localhost address and port number is :1234



* + There is command show us 3 assembly instructions starting with line we stand , the arrow points to reset symbol in startup.s file :



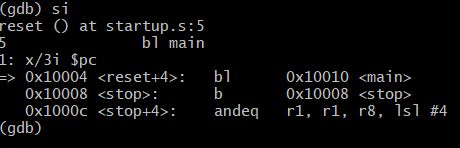
If we want to make breaking point at main The main function at address 0x10010 :



We found out that real address of main symbol is at 0x10018

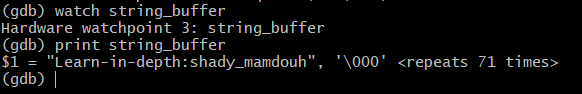
Notice : the address of 0x10010 is related with context instructions it is about creating stack and store PC in lR

* + If we want to step one instruction in assembly we can use “si” command but if we debug in C level we can use “s” command that step one C line that may contains many assembly instructions :



-if we want to print a specific variable we can use “ print var\_name “ .

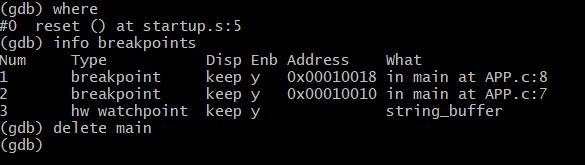
-If we want to watch a specific variable that debugger will stand if their value has been changed , we can use command “watch var\_name” :



-If we want to know where are we , we can use this command “where”

-if we want to know information about breaking points and their number we use command “info breakpoints”

-if we want to delete some breakpoint we can use “delete b\_name” :



* + If we want to tell gdb to continue till closest breaking point We can use command “c”
  + We will step in C until uart.c and we will find that the string will printed character by character on the qemo terminal :

A computer screen shot of a black screen

Description automatically generated

Makefile of lab 1 :

A screenshot of a computer program

Description automatically generated

# Lab2(Startup.s&Startup.c)

**Startup.s**

Board name : STM32f103c8t6

Notice: Entry point of this cortex-m3 based is 0x0800000 It must contain SP value of address that points to in sram

main.c :

A screenshot of a computer program

Description automatically generatedA computer screen shot of a program

Description automatically generated

Startup.s :

We gave command to assembler to make section called vectors And we defined first word as a value of SP is 0x20001000 Within range of sram

According to specs the interrupt vector table must start after SP assigning , so we make vector\_handler to handle any interrupt

A screenshot of a computer program

Description automatically generated

Linker script :

According to specs flash memory starts with 0x08000000 And sram starts with 0x20000000

-we make vector section at the start of sections to be located at the start of flash memory

A screenshot of a computer program

Description automatically generated

Make file : somethings will be edited compared with lab1 such as project name and board name :

A screenshot of a computer program

Description automatically generated

# Lab2,part2

**Startup.c**

* + As we mentioned before the reason that stop you from coding Startup.c is initializing stack because c codes use stack , so some boards have a feature allow you to initialize stack with just write the address that you want SP to point in the entry point of processor
  + Board name : STM32f103c8t6 arm-cortex-m3 based .
  + Flash starts with 0x08000000
  + Sram starts with 0x20000000
  + We want to make . text section starts with start of flash And contains . vectors section as a first section then other

.text sections from all files

* + .vectors section will contain SP and interrupt vector table So the first symbol in .vectors will be relative to the start of flash memory as we target .
  + We want to copy .data section from flash to sram and initialize .bss section in sram.
  + In linker script we will define some variables to make memory boundary at start and end of each section to help us to calculate the size of sections and to copy .data and create .bss in sram

Linker script :

A screenshot of a computer program

Description automatically generated

* + We made padding by 0x1000 memory locations in sram between .bss and stack top that will be used to create function stacks to avoid any crash .

**Starup.c :**

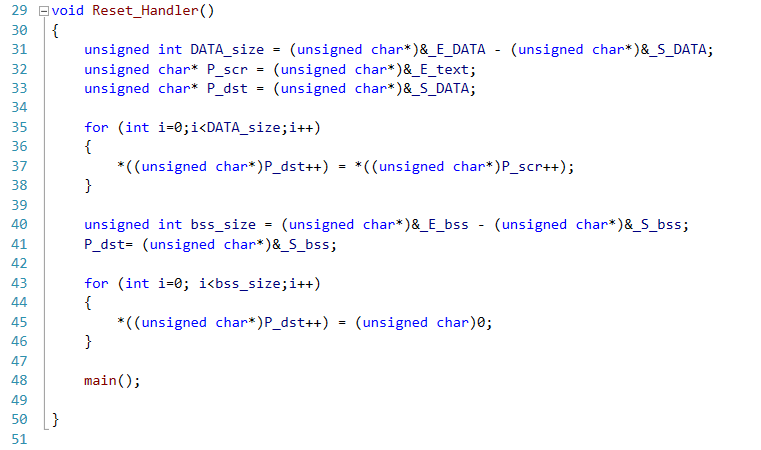
* + We use attribute to pass commands to compiler to create section called .vectors and we make array of addresses that we want to be in this section

This addresses represent SP and all interrupts vector table

* + We use attribute of weak and alias vector handler to make all vectors point to default symbol and allow user to override with his own handler

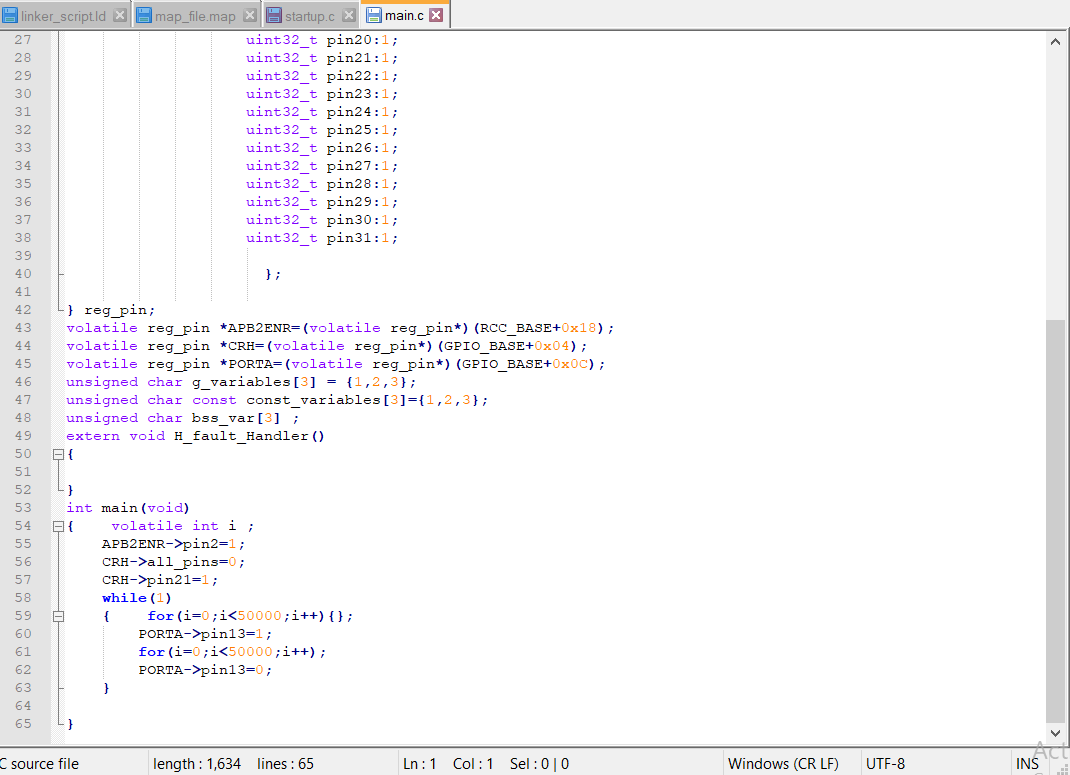
A screenshot of a computer program

Description automatically generated



**Main.c :**

* + In main we defined H\_fault\_handler() to prove concept of overriding the default symbol and change the symbol address
  + We defined uninitialized global variable to represent .bss section



* + lets make sure that everything is correct

.text section has LMA equal VMA starts with 0x08000000 As we want

Because it hasn’t been copied from flash to ram

* + .data section has LMA within flash range and it will be copied to sram so it has VMA within start of sram as we want
  + .bss section has VMA within sram range .

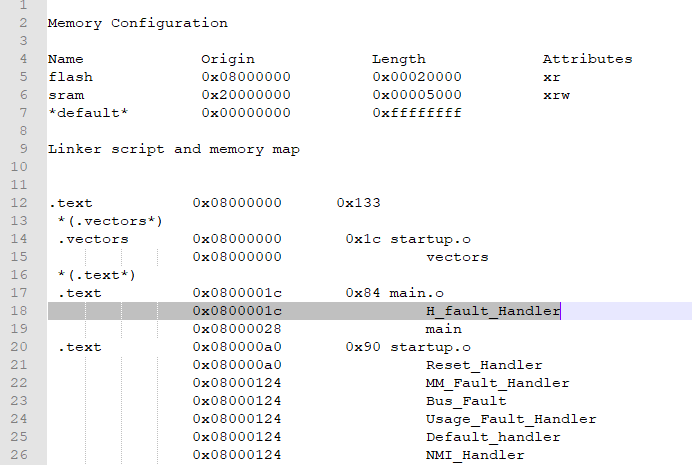
A computer screen shot of a black screen

Description automatically generated

Lets see map file to get more details :

* + H\_fault\_handler has address of 0x0800001c

That is different from the default address of other handlers 0x08000124 to prove concept of overriding

* + .vectors section at the start of flash
  + .data section has load address of 0x08000133 in flash and 0x20000000 at the start of sram as we want
  + .bss section starts with 0x20000010 and end at 0x20000013 And there is memory aligning occurred with 1 byte

