Assignment 5a
(By Vasudev Singhal Roll no 2019126)

Code Description:
Firstly we made the system start in 16 bit real mode . Then we set up the 16 bit system and set up a global descriptor table. This table helps the system to transition from 16 bit system to the 32 bit system effectively .

The boot function disbales al the interrupt so that there is no race condition , then loaded the pointer to the 32 bit global descriptor, sets all the cr0 register contents to the eax register, and then sets the first bit of the eax register to 1 . These changes made in the eax register are then saved back to the cr0 register. So now, the cr0 register contains all the correct components. Then , since we are required to use the system in a 32 bit manner, we jump to the boot32 label . This label is the key in booting up the 32 bit part of the system.

The gdt_pointer is required in order to load the table which consists of GDT size which is followed by a pointer to the gdt structure.

Lets now print the actual words required.

We need to print the words ``Hello world!" on our qemu emulator.This was done by first defining 2 strings, named str1 containing "Hello" and the second string str2 containing "world!". To print these, , we need to first load the 2 Strings in the esi register and then print its contents using the ebx register and ax vallue.

Now, we come to the printing of the contents of the cr0 register after printing the hello world string. This was implemented in the following manner:

The VGA Buffer is the most crucial element required to print the values on the qemu emulator. So , I first set up the VGA buffer. Then the edx register is set up to 31, acting majorly as a counter in counting the 32 bits and printing them . Then in our printcr0 function , the counter is checked first whether it is 0 or not and further action is taken depending on this. If its not zero , we need to keep moving in the loop ahead and print values but if it is zero, we reach the end label and terminate the program. The contents of the cr0 register into the eax register for printing in case the counter edx in not 0 . Another counter ebx is initialized to 0 for further use.

In the shift_eax function, comparison is done between the value in the ebx and the edx register.If found equal, we move to the next label, but if not, we right shift our eax register value and increase our ebx register value till they do not match.

In the 'next' label, the eax register value is ANDed by 1 firstly. Since each character takes up 2 spaces,the ecx register is incremented by 2 . eax contains the current bit(which requires printing), and we navigate to eax_one function if the current bit is 1 , else we continue on the code further if the value is zero and 0 is printed. If the value was 1, we print the value 1 in eax_one function.

To run the bootloader, we need to type in the following 2 commands:

- nasm -f bin Q1_2019126.asm -o Q1_2019126.bin
-qemu-system-x86_64 -fda Q1_2019126.bin

References:

http://3zanders.co.uk/2017/10/16/writing-a-bootloader2/
https://blog.ghaiklor.com/how-to-implement-your-own-hello-world-boot-loader-c0210ef5e74b