ASSIGNMENT 5 WRITE_UP

NAME : SUYASHI SINGHAL
SECTION - B
ROLL NO - 2019478

QUESTION 1

Description of code along with logical and implementation details

In the code we do the following things -

1) Start the bootloader in 16 bit real mode
2) Enable protected mode by setting the PE bit as 1 in the CR0 register.
3) We thus switch to the 32 bit protected mode
4) This also includes setting the values in global descriptor table
5) Print the hello world and value saved in the CR0 register indicating that we have switched to the protected mode

Now following are the functions used to achieve the same

a) **WHEN IN REAL MODE**
   We are initially in the 16 bit mode.
   Here we set the initial address to boot using the org command.
   We also clear the interrupt flag using the cli command.
   We then load the global descriptor table defined by us later.
   Then we jump to the function “protected_mode_switch” to switch to protected mode from 16 bit real mode.

b) **SWITCHING TO PROTECTED MODE**

- We basically set the PE bit of the CR0 register to 1 which indicates we have switched to the protected mode.
- This is done by first storing the value of CR0 in a register. We then set the lsb as 1 (which is the first bit or the PE bit of CR0 register). Then we put this value back into the CR0 register. Hence in this way we shift to the protected mode

c) **Enabling the Global descriptor table**
   - Here the global_desriptor is the pointer to this table
   - global_desriptor_main marks the starting of the table used to calculate it’s size.
   - global_desriptor_null is a null descriptor and part of this table
   - global_desriptor_code is the code segment. Here we set a variety of things like access, privilege, readable , writable etc.
   - global_desriptor_data is the data segment. Here too we set a variety of things for the data
   - global_desriptor_end is the end of the table. It does not contain any code but it is used for calculating the size of the table
   - We calculate the size of the table and the address of the start of the table

- We also give names to our code and data segments, namely CODE and DATA defined here.

d) In 32 bit mode

- Here, we do the actual work of printing the “Hello World” and the value of the CR0 register.
- Firstly we initialize the values of the segment registers and put the stack pointer to the top of the free space. This is done in the “initialize” function.
- We then jump the the print_values function.
- print_1 contains the “Hello World!” which is to be printed on the terminal.
- We also put the value of CR0 in the register edx to be printed.
- ecx contains the counter that starts from 0.

Printing value of CR0 register

- Here, what we are essentially doing is left rotating the register one by one and printing its value. We do this 32 times since CR0 is a 32 bit register.
- rol command is used to left rotate the register by 32 bits.
- We increase the counter ecx by 1 each time in the continue function and compare it with 32. If it is equal to 32, we move on to printing the hello world command.

check:

In the check function, we left rotate the register edx which stores value of CR0. This sets the carry flag. If the carry flag is 1, that means the bit is 1 and we go to add_1 function. Otherwise if the carry flag is 0, the bit is 0 and we go to the add_0 function.

add_1 and add_0 :

In add_1 function, we make the value of eax as 1 and print it. On the other hand in add_0, we make the eax register as 0 and print it. We then jump to continue function in both cases.

continue:

In the continue function, we increase the ecx counter by 1. We increase the value of ebx by 1 since it points to the point where we write on the qemu emulator. We compare it with 32. If it is equal to 32, then we stop and go on to print the next thing. Otherwise, we again go back to the check function and this process is repeated all over again.

Printing Hello world

Here what we are doing is printing the hello world message byte by byte(ascii value)
We moved the print_1 into esi initially.

hello:

We load a byte from esi into al. If al =0 then, we terminate the program.
Otherwise we print the message. We then increase ebx by 2 to shift the position of the offset.
When al becomes 0 the loop exits and goes to stop function.

stop:

In the stop function, the flags are interrupt flag is cleared and program in halted.

HOW TO BOOT THE BINARY IMAGE

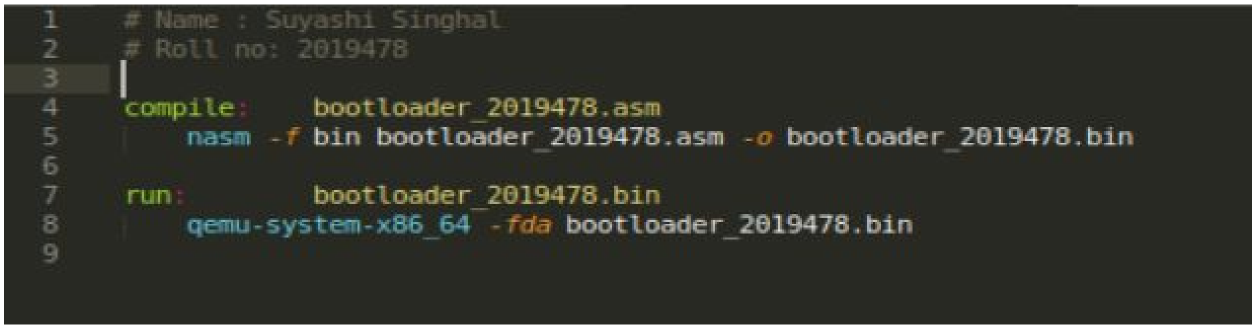
We run the following commands in the terminal using our MAKEFILE
a)**nasm -f bin bootloader_2019478.asm -o bootloader_2019478.bin**

In this command we, convert the assembly language file into the binary file to be booted

b) **qemu-system-x86_64 -fda bootloader_2019478.bin**

In this command we boot the binary file using the qemu emulator.

MAKEFILE



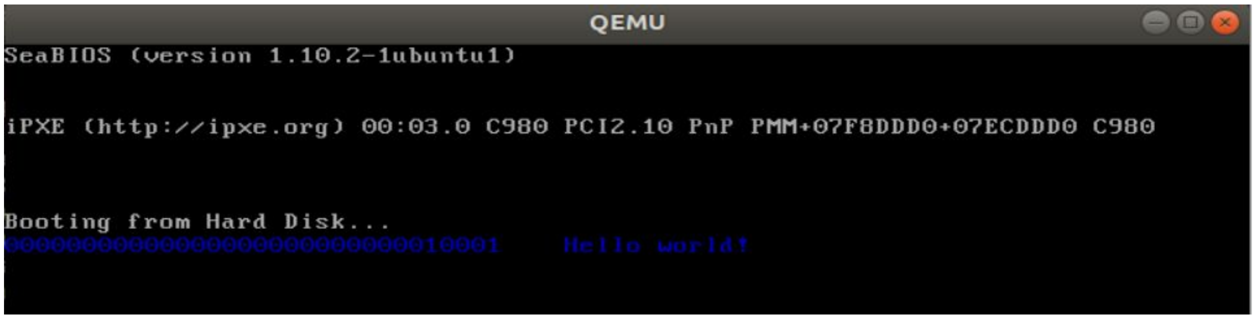
```
# Name : Suyashi Singhal
# Roll no: 2019478

compile:    bootloader_2019478.asm
    nasm -f bin bootloader_2019478.asm -o bootloader_2019478.bin

run:        bootloader_2019478.bin
    qemu-system-x86_64 -fda bootloader_2019478.bin

```

OUTPUT ON RUNNING THE BINARY



```
SeaBIOS (version 1.10.2-1ubuntu1)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDD0+07ECDDD0 C980



Booting from Hard Disk...
00000000000000000000000000010001    Hello world!
```