Phase 1 Report

1st Stage:

- **Bios_cls**
  - This function uses INT 10 for
    - Clearing the screen
    - video display mode function
- **Bios_print**
  - This function uses INT 10 for
    - Clearing the screen
  - It receives the string address that ends with a null character
  - loops on each byte
    - if it's not zero it uses displaying char function
    - repeat until zero is reached
      - ends the function and returns
- **Detect_boot_disk**
  - This function is used to detect whether we have booted from a floppy or a disk
    - This is done by checking the number that is stored in DL according to the drive table in the slides
  - If it was booted from floppy / Disk
    - Print a message on the screen
    - If it was from a Disk
      - Call a function to load the parameters of this disk
  - if the device that was booted from was unknown
    - Print a message on the screen
    - Exit with error on the screen
- **First_stage_data**
  - This function is used to declare and specify all the data that will be used in the first stage.
    - First, we define memory variables to store boot drive number, initialized to zero, and another memory variable (lba_sector) that stores the next sector to read, intialized to 0x1 since the first sector is already loaded by hardware and stored in lba 0x0.
    - We then specify the memory variables that would store the number of sectors/track and head/cylinder. Both variables are initialized to the default values of the floppy.
    - Memory variables for Cylinder, Head and Sector are also used to be able to able to use CHS.
    - Finally, we define the messages that will be used in the first stage.

- **Get_key_stroke**
    - This function uses INT 16 for:
        - Waiting to read a key press to be able to proceed and jump to second boot stage
- **Lba_2_chs**
    - This function performs the conversion from lbs to CHS
        - Make sure that dx=0
        - Move the vale of lba into register ax to be able to perform needed calculations
        - Divide ax by spt then increment remainder by 1 to get the [sector] value which is the remainder of [lba_sector]/[spt] +1
        - Divide the quotient of the previous division by hpc to get the [cylinder] value which is the Quotient of (([lba_sector]/[spt]) / [hpc])
        - Finally, [Head] is equal to the remainder of the previous division (([lba_sector]/[spt]) / [hpc])
- **Load_boot_drive_parameters**
    - This function is used to update the hpc and spt of a disk number and fetch that disk's parameters.
        - First, function uses INT 13 to fetch the disk's parameter by taking the disk number.
        - We then increment the value of the last head base zero, stored in dh by 1 to retrieve the number of hpc.
        - This number is then placed into the lower byte of hpc.
        - Extract the 6 rightmost bits of cx to extract the number of spc. Then place this number into [spc] memory variable.
- **Read_disk_sectors**
    - This function is used to read 512 sectors and store the Cylinder, Sector and Head into specific registers
        - Loop to read each disk sector
        - First convert current lba to its equivalent value of CHS by calling Lba_2_chs function
        - Use INT 13H to read each sector, one by one
        - [cylinder] of current sector is read and stored into register cx
        - [sector] of current sector is read and stored into first 6 bits of register cx
        - If the carry flag is set, jump to read_disk_error, otherwise print a dot (.) which indicates that a successful sector has been read.
        - We then proceed to read the next sector and check if last sector is reached, if not loop again.

- **first_stage (Main Program)**
    - In this main program we will include all the functions above inorder to be used
    - We will need to initialize
        - DS and SS by zero
        - Sp by the stack offset which was specified
    - Make sure the screen is cleared (Bios_cls)
        - To print greeting using (Bios_print)
    - We need to detect the disk that we have booted from using (Detect_boot_disk)
    - Then we will read the disk sectors of both the second and the third stages
    - To ensure that the second stage boot loader is loaded
        - A message will be printed and a key must be pressed to go to second boot stage
    - A hang function is implemented in the main to make sure an infinite loop occurs when an interrupt is fired

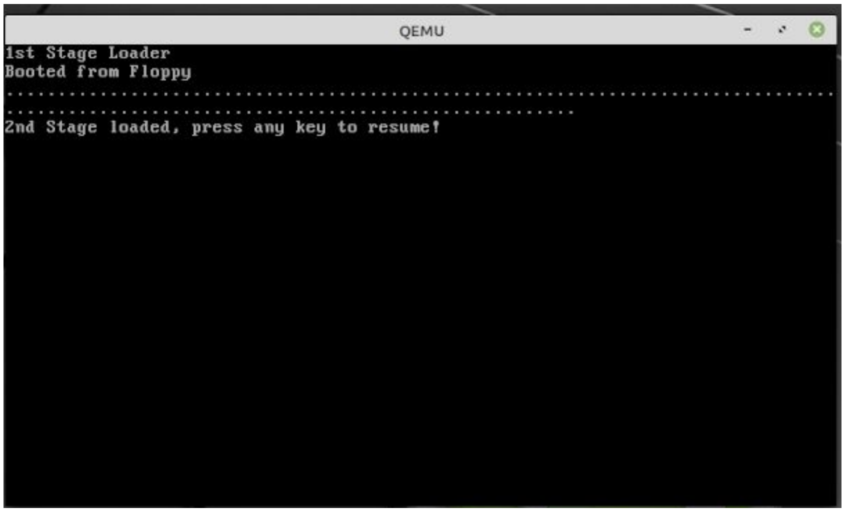
<u>2nd Stage:</u>

- **A20_gate**
    - This function uses INT 15 with function number 0x2402
        - Check if A20 Gate is enabled (i.e. is set to zero)
    - o If A20 Gate is disabled, we will enable it using INT 15 with function number 0x2401
    - o If an error has occurred, we will check AH
        - AH = 0x1 —> Keyboard Controller error
        - AH = 0x86 —> Function is not supported
        - Error is printed accordingly
- **check _long_mode**
    - In this function we want to check whether bit 21 can be overwritten or not as an indication of CPUID
        - If bit 21 is flipped then CPUID is supported
        - Else it is not supported
    - This is done by pushing the eflags in to the stack
    - Then make only bit 21 equals to 1 in eax
    - Move eax to eflags and check bit 21
- **second_stage (Main Program)**
    - Make sure the screen is cleared (Bios_cls)
        - To print greeting using (Bios_print)
    - Get a keystroke using (Get_key_stroke)

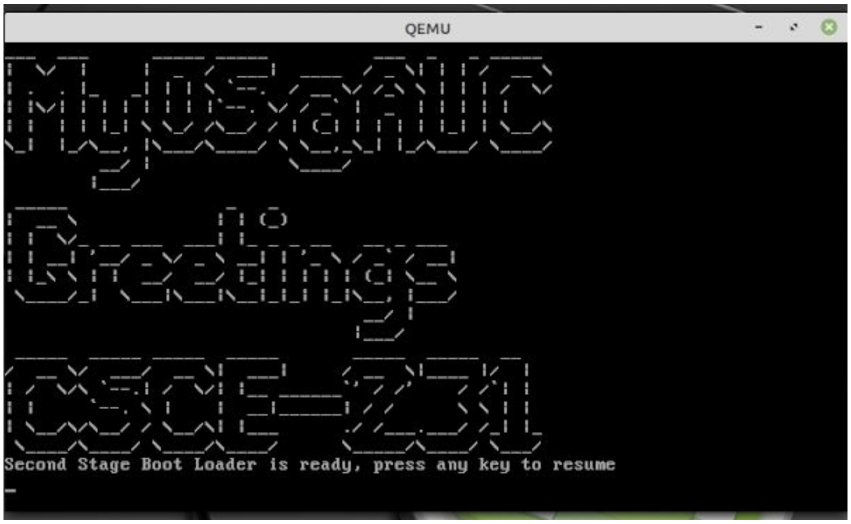
<u>3rd Stage:</u>

- **third_stage (Main Program)**

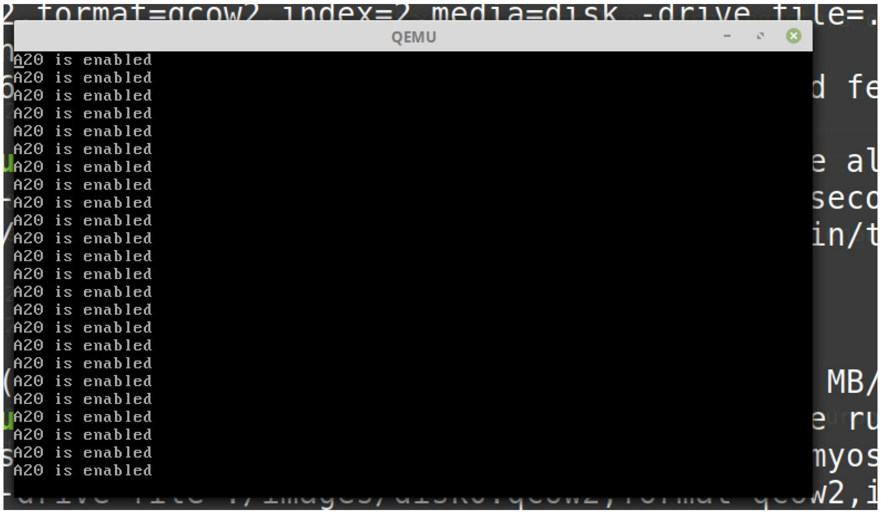
Screenshots:



```
1st Stage Loader
Booted from Floppy
................................................................................
.......................................................
2nd Stage loaded, press any key to resume!
```



```
Second Stage Boot Loader is ready, press any key to resume
_
```



```
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
A20 is enabled
```

Steps Needed to run the code:

- Open the project skeleton directory
- Open terminal in the directory
- make all
- make run_myos (for first stage run)
- make run_myos_drv (for first stage run)

Contribution: Comments and Documentation for the following Functions

Hana Asal - 900160573:

- Bios_cls
- Detect_boot_disk
- check _long_mode

Osama El Farnawany - 900161355:

- Get_key_stroke
- Load_boot_drive_parameters
- A20_gate
- First_stage_data

Salma Afifi - 900151060:

- Bios_print
- Read_disk_sectors
- Lba_2_chs