Phase 3 Report

3rd Stage:

- **Video**
  - This function was modified by adding parts that
    - clears the screen (Video_cls)
    - scrolling part within (video_print_hexa and video_print)
  - Video_cls
    - This is implemented by
      - Looping over all the cells of the video screen
      - writing in the bits for the char an empty space
      - writing in the bits for the colour 0x00 for te black colour
  - Scrolling Part
    - This was added inside the video_print and video_print_hexa by
      - Comparing the pointer that is responsible for pointing at the char that is being printed by the last possible offset that can be written on the screen (starting from the base address)
      - If the pointer exceeded this offset then the following will happen in order to implement scrolling:
        - The first row of cells in the screen will be emptied
        - All the other rows will be shifted upwards by one line
        - Then the string or hex number will be written in the last line in the screen that will be empty by now since all the rows are shifted upwards
- **PCI**
  - Get_pci_device
    - It sets the following to zeros at first
      - Bus number in bits (16-23)
      - Device number in bits (11-15)
      - Function number in bits (8-10)
    - Then setting the enable bit (bit 31) to be written later on the ports in order to enable them
  - Pci_config_space_read_loop
    - In this loop, we will read from the port (4 bytes at a time) from the header on the device
    - This header table (16 rows , 8 bytes for each row) will be copied in an empty array that is previously initialized by looping over its offset and advancing it by 4 bytes for each time we copy 4 bytes
  - print_deviceID

        - Print the device ID of the found device by extracting the device id bits from the pci header using the previously defined struct
    - print_vendorID
        - Print the vendor ID of the found device by extracting the device id bits from the pci header using the previously defined struct

➢ **PIT**

- This function was modified by adding parts that
    - Configures the PIT on mode 3
    - Defined PIT_COUNT that is 1000 (1000 pit ticks)
    - Creating a new variable print_counter that won't reset
    - Every 1000 ticks print the print_counter in hexadecimal form
    - Goes on forever

➢ **PIC**

- This function was modified by adding function clear_irq_mask
    - Identical to set_irq_mask, but inverting the bit using *not* instruction before OR-ing to unmask

➢ **ATA**

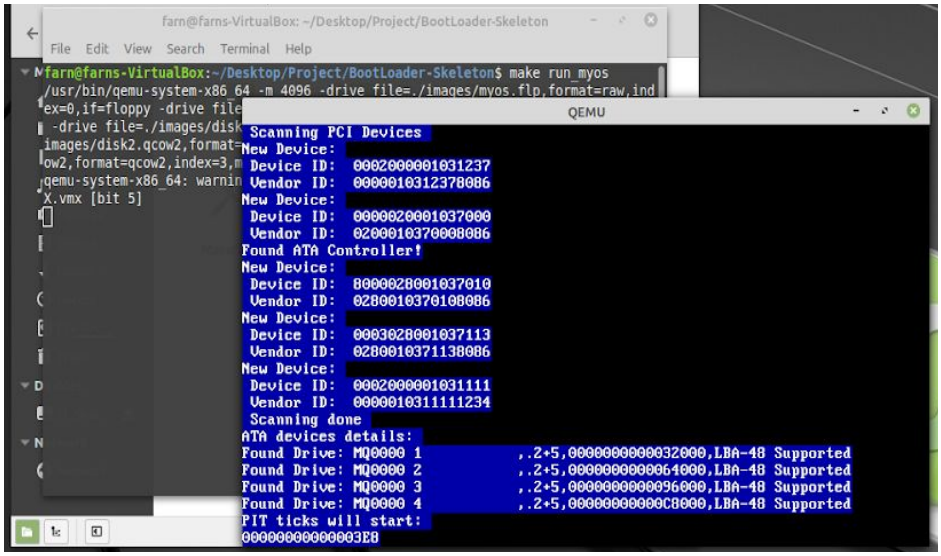
- First we define all ata ports, selectors, commands, status and offsets. Moreover, we define two memory spaces for storing the ATA Controller PCI Header (ata_pci_header) and the 4 ATA devices identify details (ata_identify_buffer) and some macro definitions, and the ATA Identify Configuration Struct for the ata attributes
- Ata_copy_pci_header:
    - This function is used to copy the info in the pci header (from get_pci_device function) into ATA Controller PCI Header
- Select_ata_disk:
    - Rdi will contain 0 or 1 based on whether we wish to write to the primary or secondary base port, and rsi will contain 0 or 1 based on master or slave.
    - We then write to the ATA_REG_HDDEVSEL offset the selector value depending on rsi (master or slave)
    - That way, in the end, we would have selected the disk we wish to work with and the ATA_REG_HDDEVSEL offset will have the correct value.
- Identifying ATA Disk:
    - Rdi will contain 0 or 1 based on whether we wish to write to the primary or secondary base port, and rsi will contain 0 or 1 based on master or slave.

        - Since for this function we only want to identify the ata disk, we write zeros in seccount0, LBA0, LBA1, LBA2 offsets and then write the IDENTIFY command in the command offset
        - We then read the status of the base port, to check for error, if no error we check if it is ready and has PIO data to transfer. If not we keep looping till port becomes ready and has PIO data to transfer.
        - When port is ready, we read from it the ATA Identify Configuration Data into the ata_identify_buffer, then proceed to print the needed attributes/info by calling the "ata_print_info" function
    - Ata_print_info:
        - This function is used to print the attributes/ info of the ata device using the offsets defined in the ATA Identify Configuration Struct to get the specific data I need.
    - Read_disk_sectors:
        - In this function, we have calculated the sector address that we need to start reading from, in the drive (137)
        - We have calculated the number of sectors that we need to read
        - We will need to loop for 10 times and read 256 sectors at a time
        - We have started looping on the port to read from it word by word until we have reached (total number of sectors to read *256)
            - Each time we read a word it is stored in memory starting from a defined memory address that its offset is being incremented each time by 2 bytes for the next time we read
    - Database_print_test:
        - In this function, we have tried to print part of the database that was stored in the memory ; to make sure that the database was copied correctly to the memory before scanning it to search for the device ID and Vendor ID of the found devices
    -

➢ **third_stage (Main Program)**
    - Added the 3 nested loops for the PCI (bus, device and function)
    - Added a loop to print the PCI scanned devices and a message if an ATA controller was found
    - The sequence of our main is as follows:
        - Calling page table function
        - Scanning PCI devices
        - Scanning ATA devices
        - Initializing and setup IDT

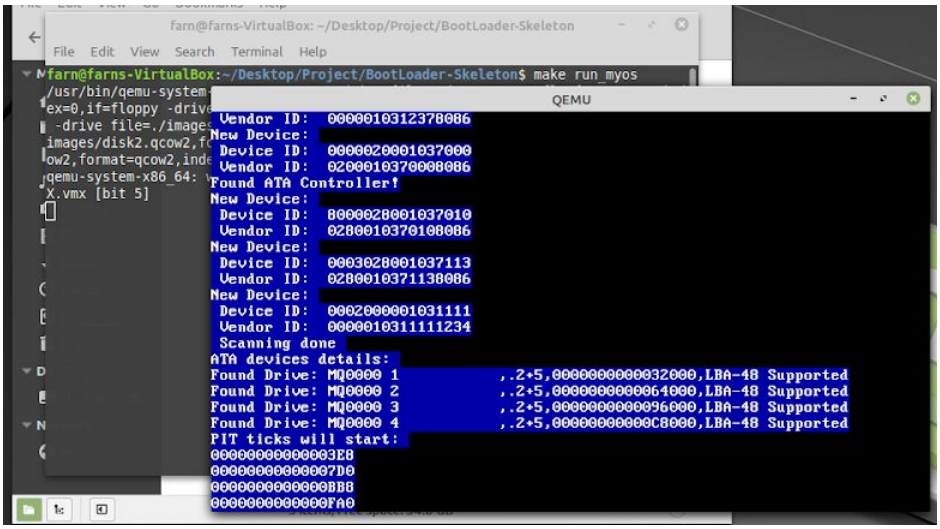
- Added couple of messages and video clears for better output view

Screenshots:

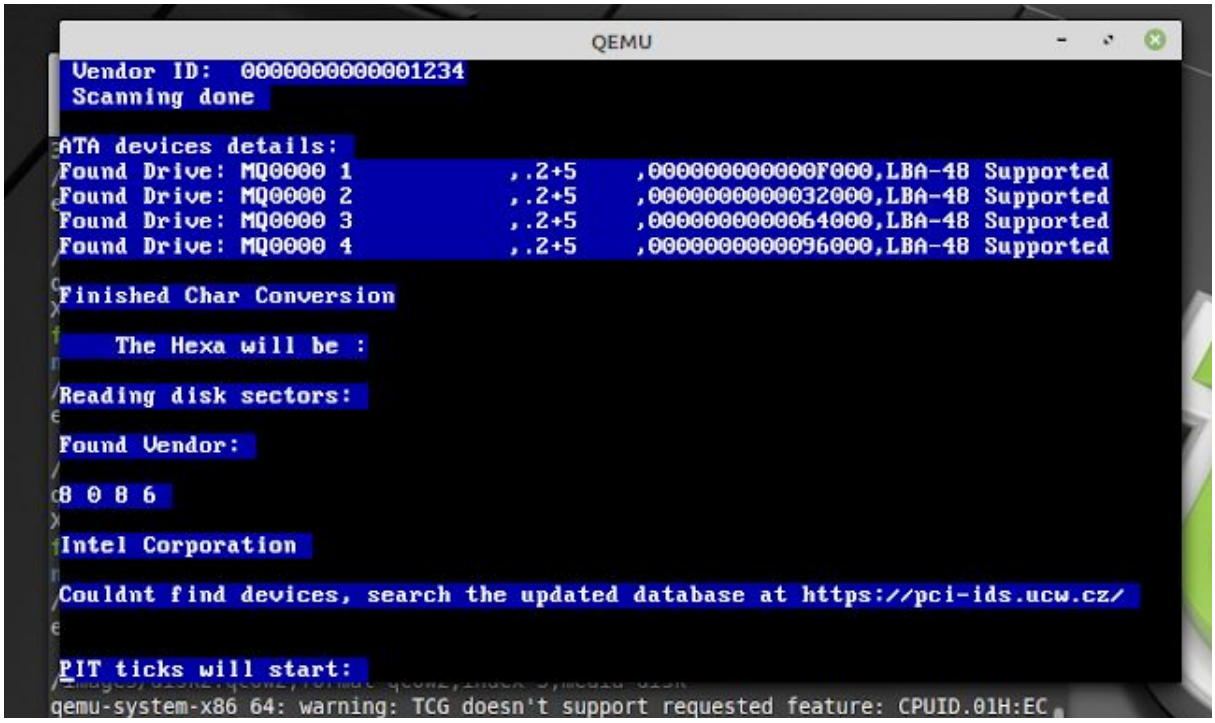


(Scanning PCI devices and showing ATA device attributes)

(showing scrolling)



(showing read disk sectors)



Steps Needed to run the code:

- Open the project skeleton directory

- Open terminal in the directory
- make all
- make run_myos

Contribution: Comments and Documentation for the following Functions

- All this phase's work was done together, even the documentation on the skeleton's code wasn't done by anyone of us separately.