Phase 2 Report

2nd Stage:

- **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
- **GDT**
  - This is a function that will represent the start base address, limit, access attributes and flags of the following:
    - Null Descriptor → unusable
      - Everything is set to zeros
    - Kernel Code Descriptor
      - The access bytes are set
        - Present bit
        - Privilege bits (Ring 0)
        - Executable bit
    - Kernel Data Descriptor
      - The access bytes are set
        - Present bit
        - Privilege bits (Ring 0)
        - Readable/Writable bit
        - Access bit
- **IDT**
  - This function is used to initialize and load the IDT descriptor, which is just a data structure holding the size and address of the IDT
  - It makes sure that the IDT is starting at a 4-byte aligned address

➢ **Long Mode**

- CR4 is configured by setting
  - Bit 5 (Physical Address Extension)
    - Translation of virtual to physical addresses
  - Bit 7 (Page Global Enabled)
    - Translated addresses are shared between different address spaces
- CR3 is configured after mapping the 2mb Page Table by pointing it to PML4
- EFER is configured by setting
  - Bit 8
    - Enable Long Mode
- CR0 is configured by setting
  - Bit 0
    - Protected Mode (NO Interrupts)
  - Bit 31
    - Paging is enabled to switch to long mode
- GDT is loaded then we'll go to the third stage (long Mode 64 bit)

➢ **Memory Scanner**

- In this Function, we use INT 0x15/Fn0xe820 to read all memory regions which expects certain value to be stored in eax, ebx, es:di, ecx and edx.
  - Value at address 0x2000:0x0000 is used as a counter and initialized to 0 to store the number of memory regions
  - The table in which the regions and the info about it will be stored (Memory Region Table) is at address 0x0534D4150
  - We then start looping to scan all memory regions
  - After we INT 0x15 is initiated, we check the value in register eax, if it contains the 'Magic Number; 0x0534D4150, the region was scanned successfully
  - We then proceed to the next entry in the Memory Region Table to store in it the region that will be read in the next loop
  - Counter of the memory regions at address 0x2000:0x0000 is also incremented.
  - If ebx reaches 0, we exit the loop as INT 0x15 uses ebx as an index and makes it 0 again when all regions are scanned.

➢ **Page Table**

- In this function, we need only to map 2 mb to allow us to switch to long mode
  - **PML4**
    - The page is created at the address [0x1000:0x000]

        - Then a pointer from PML4 is set to point at the address of the lower level page (PDP)
    - **PDP**
        - The page is created at the address [0x1000:0x1000]
        - Then a pointer from PDP is set to point at the address of the lower level page (PD)
    - **PD**
        - The page is created at the address [0x1000:0x2000]
        - Then a pointer from PD is set to point at the address of the lower level page (PTE)
    - **PTE**
        - The page is created at the address [0x1000:0x3000]
        - This is the last level of the page table that maps directly from physical memory
        - .pte_loop is the loop used to map the first 2mb from physical memory

➢ **PIC**

- Function of this is to code the microcontroller (PIC) by using the ports to disable it, as this is a crucial step to switch to long mode
    - 0xFF is written on both Master and Slave ports to disable them
    - 2 nops (No Operation) are needed to give time for the ports to shut down
    - Printing a message indicating that the process succeeded

➢ **Video**

- Function of this file is to clear the screen before switching to long mode and starting the third stage
    - Address of the VGA cannot be used directly as we are still in the 16-bit (Protected) mode, so augmenting es:di was done initially
    - A loop to print spaces (‘ ‘) in all the screen with a black background and stopping at 0x0FA0

➢ **second_stage (Main Program)**

- Calling all the functions needed before going to third stage
- Calling the video_cls_16 to clear the screen
- Jumping to the address of the third stage (0x10000)

<u>3rd Stage:</u>

➢ **Page Table**

- This is the full page table to map all memory of type 1 to be used later
- Contains main function *page_table* and another two *get_max* and *check_region*
- Defines:
  - Four addresses of the four levels
  - The address preceding the PTE level, at 0x104000
  - Constants used throughout the file
  - Location of the Memory Regions table taken from memory_scanner function
  - Counter of the Memory Regions table taken from memory_scanner function

- Variables:
  - Four pointers to the four addresses defined of the four levels
  - Pointer to the location after the pte (AFTER_PTE) at 0x104000
  - Pointer to the physical memory starting at 0x0000
  - Variable to have the maximum address of type 1 (max) to be used to hold the result of get_max function
- get_max function :
  - Used to get the address of the maximum memory of type 1 that can be mapped, in order to know when to stop mapping
  - Uses the location of Memory Region table as well as the counter taken from memory_scanner function
  - A loop on the table's rows that starts with the counter taken from memory_scanner, will first hold the start and the end (range of the addresses) in temporary registers
  - Then by adding 16 to the start, will extract the type then compare it with 0x1 to know whether its of type 1 or not
  - If it was type1, then hold this address into the variable max
- check_region function/label:
  - Compares the physical address at hand with the start and end addresses in a given range, to make sure it's in a correct region
  - If it was between the boundary then go check it's type (whether it's type1 or not)
  - If it was type1 then go to the label "type1" in the main function *page_table*, and if not then go to "continue1"

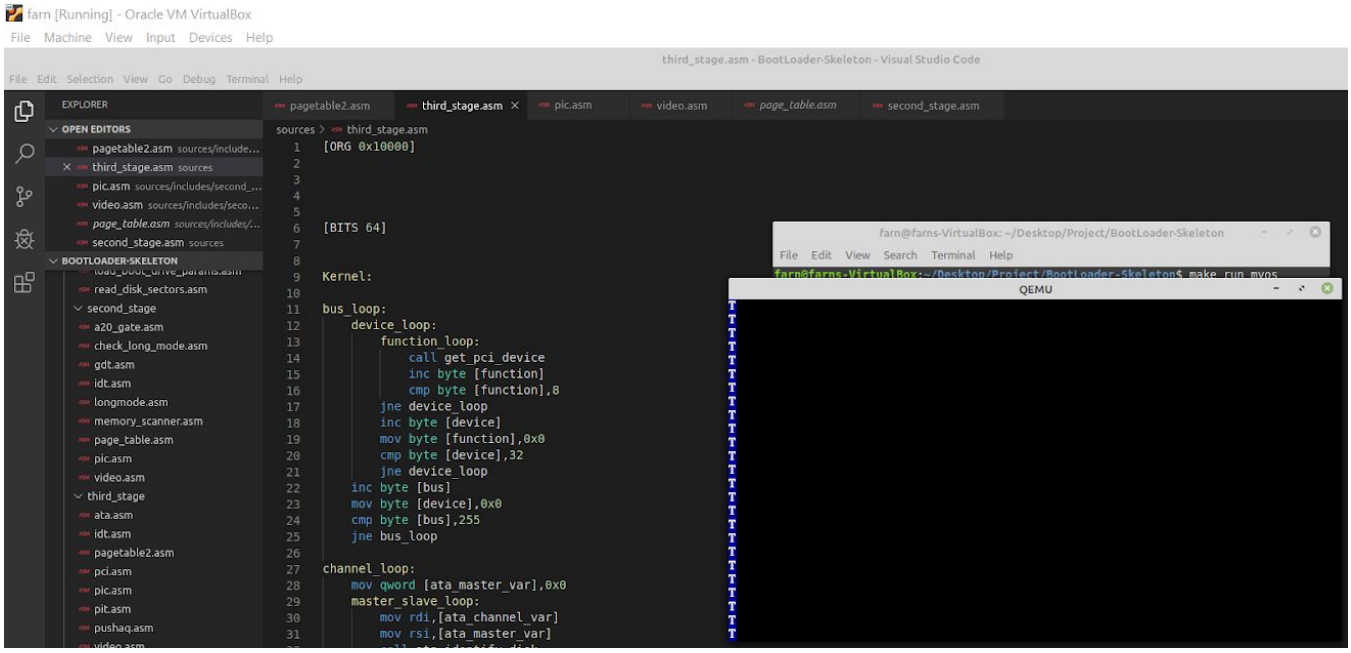
- page_table function :
  - Made of four nested loops for the four levels
  - Call get_max function before entering the loops to know the maximum to stop at
  - **PML4 loop:**
    - Used to create 512 PDP tables (level 3)
    - Increment 1 table cell (8 bytes) each time we enter the loop and point this cell to a new table of PDP
  - **PDP loop:**
    - Used to create 512 PDT tables (level 2)
    - Increment 1 table cell (8 bytes) each time we enter the loop and point this cell to a new table of PDT
  - **PDT loop:**
    - Used to create 512 PTE tables (level 1)
    - An update for CR3 register with location of PML4 is done here in order for the CPU to see the new tables that maps the physical memory
    - Increment 1 table cell (8 bytes) each time we enter the loop and point this cell to a new table of PTE
  - **PTE loop**
    - Used to map 2mb of physical memory
    - Each time the loop is entered a check will be done to see whether max was reached or not, and if yes; then exit the whole function
    - Each time the loop is entered compare the physical memory address at hand with 0x10000 (1mb), if it was less, then don't check its region and type. The reason for this is that the first 1mb contains stuff with many regions other than the first, like the VGA or the BIOS code; and map it directly.

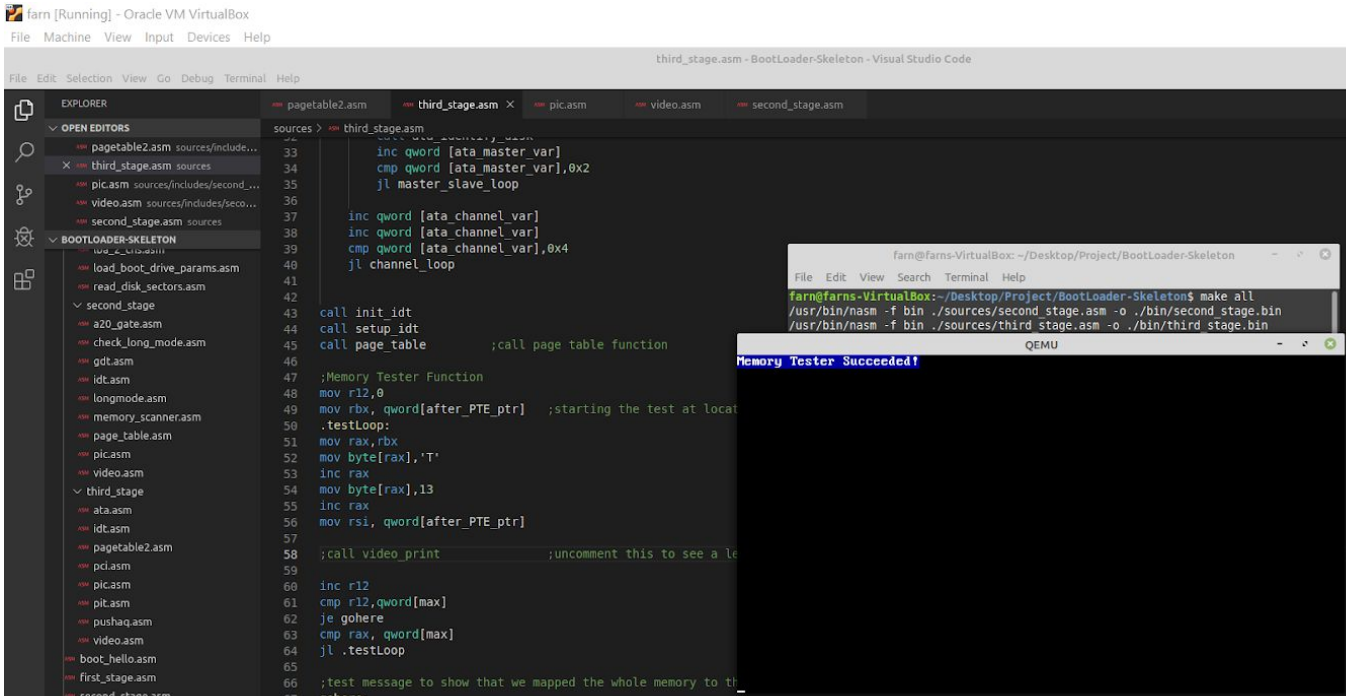
  - At the exit label, CR3 register should be updated by PML4 for one last time for the CPU to see the changes

➢ **third_stage (Main Program)**

- Used to call the functions initializing IDT, setupping IDT and building the full page table to map the whole memory
- Contains the memory tester loop to test the whole memory and to check that there's no page faults

Screenshots:





Steps Needed to run the code:

- Open the project skeleton directory
- Open terminal in the directory
- make all
- make run_myos
- The run takes a couple of minutes and the cursor keeps blinking and will output the letter T on consecutive lines
- For testing the whole memory, comment line 57 in file “third_stage.asm” to show only that a message that we finished inputting and outputting the letter T on the whole memory region.

Contribution:

Page Table in Third Stage and **Video** in Second Stage, were done as a group, and we all contributed.

Comments and Documentation for the following Functions:

Hana Asal - 900160573:

- Page Table (2nd Stage)
- GDT

Osama El Farnawany - 900161355:

- Long Mode
- PIC

Salma Afifi - 900151060:

- Memory Scanner
- IDT