Assignment 5

Name: Vaibhav Girish
Roll number: 2019121

I implemented a simple bootloader, which was booted via a legacy BIOS.
The program was written in assembly language and a bootable binary was created. This bootable binary would first switch to the protected mode (32-bits) and thereafter print “Hello world!” and the contents of the cr0 register.
I booted my bootable image using qemu emulator.

The code for the above in assembly is saved in the file boot.asm
I began by first declaring that the code started with 16-bit instructions and a memory base address of 0x7c00.
Now we need to enter the Protected mode and the addresses will no longer be linear.
We can handle memory access in two ways Segmented or Paged.
For this code I used the segmented memory model.
To set up our segments we must first create and load the Global Description Table (GDT). I created two segments, the code segment and the data segment.

We must first set the **null segment** which is reserved by intel and we set it 0

In the code segment:

For the 1st Double word (Bits 0 - 32 [dw instructions]):
The first 16 bits (Bits 0 - 15) sets the limit of the segment to 0xFFFF (aims for 4GB)
The next 16 bits (Bits 16 - 31) sets the base address to 0

For the 2nd Double word (db instructions):
First 8 bits (Bits 0 - 7): Continues on the base address so it is set to 0
The next 8 bits (Bits 8 - 15) : set to 10011010 (15 - 8 from left to right respectively)

I. Bit 8 - access flag set to 0 (will not be used)
II. Bit 9 - Sets the segment to be readable
III. Bit 10 - Conforming bit (less privileged code segments are allowed to call this segment) set it to 0 since we don’t want conforming in an OS.
IV. Bit 11 - Executable bit, if 1 code can be executed
V. Bit 12 - Specifies if its a code or data segment hence set to 1 should be 0 for system segments

VI. Bits 13 - 14 : Sets the privilege or ring level , since this is the OS it must be 0
VII. Bit 15 - Present flag set to 1 for valid selectors

The next 16 bits (Bits 16-23):

I. Bits 16-19 : Last bits in the segment limit
II. Bit 20 - Flag which is ignored by CPU
III. Bit 21 - Reserved by intel and must be set to 0
IV. Bit 22 - Size bit 0 defines 16 bit and 1 defines 32 bit
V. Bit 23 - Granularity if 1 the limit is in 4kb blocks

The last 8 bits are still the base and are therefore set to 0

In the data segment:

1st Double word remains the same as the code segment

For the 2nd double word:
The first 8 bits still remain 0 as in the code segment
In the next 8 bits its set to 10010010
The differences in these and the code segment are:

I. Bit 9 enables write access instead of read access
II. Bit 10 handles expand direction, expansion happens downwards so it should be set to 0
III. Bit 11 we want a data segment and therefore must not be set

The last 16 bits still remain the same therefore we set them the same way as the code segment

We the write the gdt_desc which sets the the descriptor of the GDT
The CODE_SEG and DATA_SEG are the offsets in the gdt being used for the the respective two segments

Now that we have the GDT we must now enter the protected mode.
In the boot section of my code, I first enable the A20 line which gives access to more than 1MB memory at a time and enter the vga text mode.
After that I disabled interrupts so that only the bootloader gets to execute.
Then I load the gdt with the gdt_desc
And I set the 0 bit in cr0 register to 1 so that we enter protected mode

After this I jumped to my 32 bit instructions where I printed the string “Hello World!” followed by the contents of the cr0 register printed byte by byte
After that I enabled interrupts again and padded the remaining of the boot sector with zeroes and set the standard pc signature

The output was as follows

References:

1. https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html
2. http://www.osdever.net/tutorials/view/the-world-of-protected-mode
3. https://medium.com/bugbountywriteup/writing-a-bootloader-931da062f25b