Even if it is a 32- or 64-bit x86 processor, at boot time the processor will run in the 16-bit real mode, due to which initially the line [bits 16] is written to tell the assembler that we are doing it in 16 bits. To make it bootable we need 512 bytes and if the last two bytes is the magic number then the bootloader is done due to which the last line of the code fills the first 510 bytes with 1.

It can be filled with any bit 0 or 1 as what matters is the last bytes. Now to enable 32-bit instructions and access to the full bit 32 registers by entering Protected Mode, Global Descriptor Table will have to be set up which will define a 32-bit code segment, load it with the lgdt instruction then do a long jump to that code segment. To actually load Global Descriptor Table, we also need a gdt pointer structure. This is a 16- bit field containing the GDT size followed by a 32-bit pointer to the structure itself.

CODESEG and DATA-SEG are offsets for the gdt. This helps to get into the 32-bit protected mode. Now we need to print the contents of the register CR0 which is used for switching on to the protected mode and its right most bit should be 1. We also need to print “Hello world!”.

To print both the contents I have used a loop, its explanation is as follows. The top byte defines the character colour in the buffer as an int value from 0-15 with 0 = black, 1 = blue and 15 = white. The bottom byte defines an ASCII code point. The colour that is used for printing is defined to be blue here. To print the contents of cr0 register loop1 is used and for “Hello World!” loop is used. To avoid overwriting on the qemu emulator, first the contents of cr0 is printed and then after having two spaces the next the hello world statement is printed.

For the whole implementation I have referred to the following links:
http://3zanders.co.uk/2017/10/16/writing-a-bootloader2/

https://medium.com/@g33konaut/writing-an-x86-hello-world-boot-loader-with-assembly-3e4c5bdd96cf