Assignment 1

This is the first of a multi-part assignment to build SimpleOS on a 32-bit x86 machine from source. The subsequent SimpleOS assignments build on this assignment. So, you will need to have this assignment fully working to attempt the second assignment, and so on. Turning in assignments late will hurt your ability to stay on track.

Using the base code I provided in Github, as well as the links below, you will need to complete the multi-stage bootloader code. I consider this assignment a warm-up for later, more complex SimpleOS assignments, so be sure to completely understand how this code works before moving on.

https://en.wikipedia.org/wiki/BIOS_interrupt_call

https://en.wikipedia.org/wiki/VGA_text_mode

There are several parts you must finish to make the code work:

1. Using the BIOS interrupt link above, load the bootloader-stage2 binary data (i.e., the entire ELF file from fs.img) to RAM starting at address 0x8000. This will need to be split into two halves (each of 128 sectors) since we cannot load more than 128 sectors reliably with BIOS interrupts in one operation. *(This allows our bootloader-stage2 binary to grow to 128 KB and still be loaded correctly. It is possible, since this binary is small initially, that you can incorrectly load the second half and it still works (for now) but you will lose points when I grade it.)* For the first half, you will load 128 sectors of 512 bytes from fs.img to memory address 0x8000. The second half will load an additional 128 sectors of 512 bytes from fs.img to memory address 0x18000. (Hint: due to segmentation in x86 real mode, you will need to use the "es" segment register to assist as you cannot just load 0x18000 into the bx register.) Your implementation of how to load these sectors using BIOS interrupts will be placed in bootloader-stage1.asm in the locations I specified in the lecture video (the TO DO sections).

2. Thoroughly read the VGA text mode link (above). Using that information, create your implementation of clearScreen(), printCharacter(), printString(), and printHexNumber() in screen.cpp based on the name of those functions and the arguments required. **You cannot modify any of the function signatures or anything else in the files I supplied other than the function implementations.** (Additional Hints: (1) Your BIOS interrupt code should start from sector 2 and use cylinder 0 and head 0. (2) Your clearScreen() function should zero out all of video RAM. (3) Your printCharacter() function should handle Line Feeds correctly (0x0A), Tabs correctly (0x09), and should consume, but ignore 0x0D). (4) Your printString() function should print on a newline for both 0x0A and 0x0D bytes.)

3. Be sure to “make” your solution and see that it matches my screenshot below. **You cannot modify the Makefile, either.**

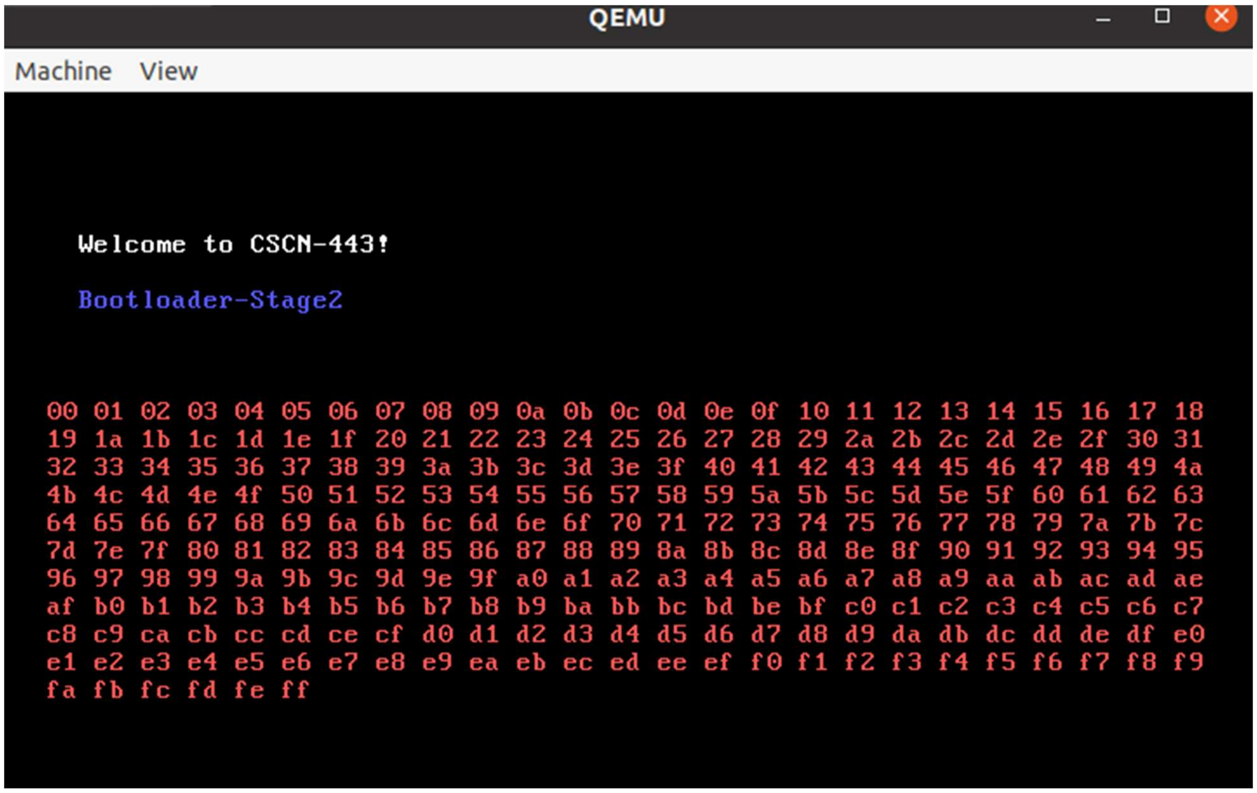
Additionally, be sure to use the constants (from constants.h) whenever possible. You should not be using a hard-coded numeric value if there is a corresponding constant in constants.h that you could use (but do not modify, add to, or alter constants.h in any way). You cannot create new helper functions and you should not create any new global variables. **Be sure to follow a professional coding standard, including the naming of your variables, structures, etc. You should use descriptive names that are appropriate. Nonsensical or whimsical names will lose points. You must also use the custom types where I have supplied them (e.g., uint8_t, uint16_t, uint32_t) unless you need a signed typed or one I haven’t supplied. You will lose points if you don’t follow this convention.**

Once complete, your solution should exactly match my screenshot below. When you feel everything is working, submit **ONLY the SUBMIT-ME.zip** file to Canvas for this assignment and nothing else. I will download and test your solution on my QEMU system. It will need to work on my computer to receive credit, so I HIGHLY recommend you use the environment I show in class.

Additionally, you must attest and add the following honor statement to your Canvas submission (in the Canvas submission comments):

I, <state your name>, attest that this submission is my own work. I have not worked with anyone else on my implementation and have not used AI or other unpermitted sources to help me with my solution, in accordance with our definition of class cheating shown in Lecture 1.

Any submission that does not include this attestation will not receive any credit.



```
Welcome to CSCN-443!

Bootloader-Stage2


00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18
19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31
32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45 46 47 48 49 4a
4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63
64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a 7b 7c
7d 7e 7f 80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91 92 93 94 95
96 97 98 99 9a 9b 9c 9d 9e 9f a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae
af b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf c0 c1 c2 c3 c4 c5 c6 c7
c8 c9 ca cb cc cd ce cf d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df e0
e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
fa fb fc fd fe ff
```