CSE231: Operating Systems

Assignment 5, Question 1

Name: Mihir Chaturvedi
Roll no.: 2019061
Section: B
Branch: CSE

For this question of the assignment we attempt to write a bootstrap program that will be employed by the QEMU emulator. We run our program in 32-bit protected mode, and print the words "`Hello, world!`" and the bit-sequence of the `CR0` register onto the video output.

Description

- The program is divided into two sections: 16-bit Real Mode code, and 32-bit Protected Mode code.
- In the first lines we specify the mode of the first section of the code, and also the starting address of the program in the bootloader.
- `init_vga`: We set up the VGA configuration (used to properly print text to screen) by initializing with a known value (3) and making a boot interrupt call.
- `switch_to_pm`: This subroutine runs the essential code required to switch interpretation of instructions from 16 bit mode to 32 bit mode.
    - We load the Global Descriptor Table, and all the addresses that are pointed by it, including those of the different segments (in its 32-bit addressing space):
        - NULL descriptor
        - Code descriptor
        - Data descriptor

    - We enable the protected mode by assigning the protected mode bit (LSB) of the `CR0` register as `1`. This is done by OR'ing the `CR0` register with 1 (indirectly).
- We then jump straight to init_pm subroutine, in the 32-bit section of the program.
    - Here, we move the "`Hello, world`" message into the `ESI` register
    - Also, we set and store the starting address of the video output in the `EBX` register. This is hardcoded to `0xb8000`.
- We continue now to print the characters of the message in a loop.
    - We load the value of `ESI` into the `AL` register through the `lodsb` operand.
    - We check whether the character is nullish (zero). If not,
        - We give the text output a white color (by OR'ing with `0x0F00`)
        - Copy the character into the video output address space (which finally prints to screen)
        - Shift `EBX` by two bytes for the next character
        - Continue the loop
    - If the character is nullish, it means we have finished printing the message, and so we break out of the loop and jump to the print_cr0 subroutine
- In the `print_cr0` subroutine:
    - Here, we will attempt to print the bit-sequence of the `CR0` register (binary)
    - We first print out onto the screen a SPACE character
    - Then, we set up the iterator variable for the 32-time loop, and copy the contents of the `CR0` into the `EDX` register.
    - We continue into the `print_cr0_loop` subroutine
- In the `print_cr0_loop` subroutine
    - We left rotate-shift the `EDX` register storing `CR0` and use the LSB.
    - We print the LSB (0 or 1) onto the screen by adding 48 to it (to convert it into the ASCII character of the digit).
    - Then, we finally check whether the iterator variable has reached 32 or not, indicating that it has rotate-shifted the entire `EDX` register fully. If yes, we move on.
- Finally, we halt the program safely.
- We pad the data section with zero's to indicate no data.
- Finally, we indicate that this sector containing the program is bootable by defining the magic number `0xAAFF`.

Instructions to Run

- To build the binary from the x86 assembly file, we will need the NASM assembly linker and compiler.
- Once we have built the binary, we will require the QEMU emulation program to load the binary as the bootloader program.
- To make these steps and commands simpler, one can simple run `make run` after they have the requisite programs installed.