3.2

- Returning From x86 Interrupts: information output before terminating
    - Xinu trap!
    - exception 6 (invalid opcode) currpid 2 (Startup process)
    - CS EFD0008 eip 102945
    - eflags 10202
    - register dump:
    - eax 00000001 (1)
    - ecx 00102781 (1058689)
    - edx 001187EF (1148911)
    - ebx 00121000 (1183744)
    - esp 0EFD8FD0 (251498448)
    - ebp 0EFD8FD0 (251498448)
    - esi 00000000 (0)
    - edi 00000000 (0)
    -
    -
    - panic: Trap processing complete...
- Testing for infinite loop
    - I created a new function called testy() in main.c that contains a kprintf() statement that prints out “Loop”. Right before I call iret in _Xint6, I call testy(). After the “movl %ebx, %cr1” instruction is executed, “Loop” is printed infinitely, meaning there is an infinite loop if the only instruction in _Xint6 is iret.
- Implementing new handler _Xint6i that returns to the next instruction:
    - Although executing sti allows hardware interrupts to occur, it does not allow software interrupts to occur. Since asm(“int $6”) is a software interrupt, it will still be disabled after executing sti.
    - Disable() blocks all IRQ exceptions, regardless if they are hardware or software interrupts. Restore() restores all IRQ exceptions to the state they were in before calling disable(). Unlike the sti instruction, disable() and restore() can handle both software and hardware interrupts. Calling these two functions in opcodeinv() works because these functions are not limited by whether the interrupt is a software or a hardware interrupt.

3.3

- When testing my _Xint6, I first tried an offset of 4. I knew 4 was an incorrect offset because when I tested my code, it would result in a Xinu trap and a register dump. I then tried an offset of 3 which I know worked because _Xint6 did not result in a Xinu trap or a register dump. To test my implementation, I followed the example in the lab handout on top of checking for Xinu traps and register dumps. I set a variable, x, equal to 0, executed movl, then set it equal to 3. I then called kprintf to print out the value of x, which was 3.

4

- Files added: clkticks.c

- Files modified: clkinit.c, clkhandler.c, clock.h