1. Bus arbitration
    a. How to arbitrate a bus
        i. Who gets the bus when?
    b. Methods of bus arbitration
        i. Centralized – bus controller allocates time on bus
            1. Bus Request Line – OR all requests from devices on bus
            2. Bus Grant Line – daisy chain across all devices
                a. Order of devices on line determines priority
        ii. Distributed – each module contains logic to access bus, work together to share bus
            1. Arbitration Line – bus is being granted right now
            2. Busy Line – bus is being used
            3. Bus Request – if another device wants to use the bus
                a. Conflicts settled by priority or random back-off
                b. Random back-off – each device waits a random amount of time before requesting again
    c. Timing mechanisms
        i. Synchronous
            1. Actions take place at specific clock cycles
            2. Simpler to implement and test, but less flexible
        ii. Asynchronous
            1. Occurrence of an event follows and depends on occurrence of previous event
                a. CPU waits for ACK from memory before sending next command
            2. Allows for both slow and fast devices
                a. Easier to upgrade, but means more logic

2. Interrupts
    a. Will be covered again in ECS 150
    b. Why do we need interrupts?
        i. OS handles the interface between internal and external portions of machine
        ii. Large speed disparity between CPU and other I/O devices
            1. Keyboard – 100 ms
            2. Disk drive – 10 ms
            3. CPU – 1 ns
    c. How can we deal with I/O?
        i. Busy waiting – OS sits in loop, waiting for key to be pressed
            1. Instant response, but must keep checking all the time
        ii. Polling – OS checks with device every now and then
            1. Less wasted CPU time, but less responsive
        iii. Interrupt – change in program flow generated by external or internal event
            1. Imagine getting a notification on your phone
            2. Type of notification determines how you respond to it
    d. What do we need to implement an interrupt?
        i. Must preserve current state
            1. Need to come back to this place later
        ii. Jump to the correct interrupt service routine / subroutine (ISR) based on the interrupt type
            1. ISR handles the interrupt
        iii. Interrupt needs to be invisible, so current state can be restored correctly
            1. Like the interrupt never happened

e. Changes we need to make to support interrupts (incomplete list)
    i. Modify our original program order of Fetch, Decode, Execute
        1. Add Check for interrupts to the beginning or end (CFDE or FDEC)
    ii. Add a place in memory to store the ISR code / instructions
        1. Need to protect this place in memory from being modified by any user processes
        2. ISRs tend to be privileged to handle data from hard drive, caches, so on
            a. If not protected, malicious programs can modify ISR
            b. Modified code would run any time interrupt is handled by OS
    iii. Support different types of interrupts
        1. Have different interrupts for keyboard versus hard drive
    iv. Need to be able to enable and disable interrupts
        1. What happens if we keep getting interrupted when handling an interrupt?
        2. Would never get anything done, would keep getting interrupted
    v. Need to know what to load into PC
        1. Interrupt changes program flow, as mentioned earlier
    vi. Need to add an Interrupt Service Routine (ISR)
        1. Routine is the address that gets loaded into the PC
        2. Handles the interrupt
    vii. Need a Return from Interrupt (RTI) instruction
        1. Once ISR is done, restore original state and go back to where we left off