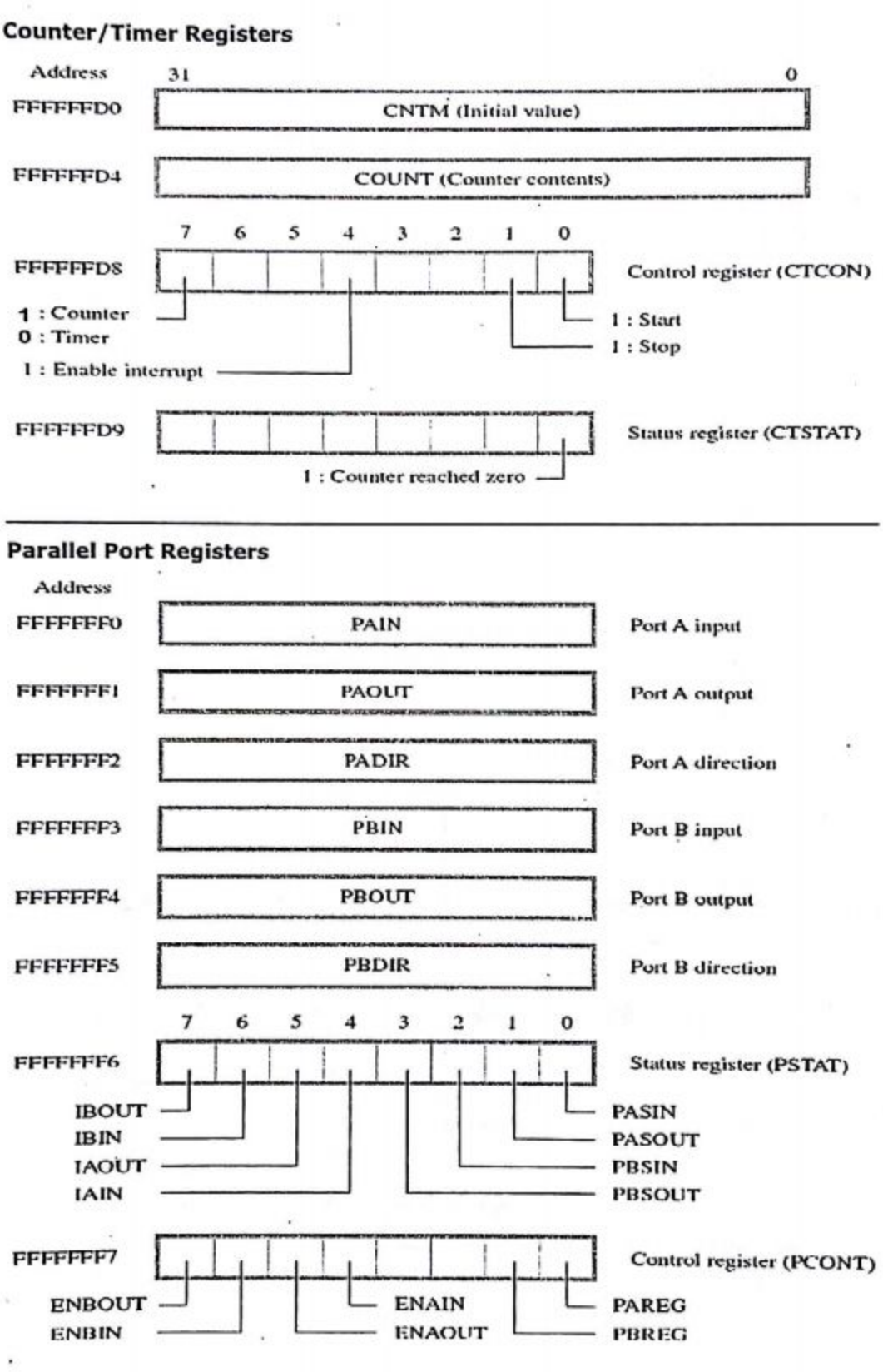
Question Type 1



Here is a copy of the registers that I believe we will be getting with the exam anyway, but they're useful for practicing questions, and good to have just in case we're expected to bring our own.

Here are some snippets of code that appear consistently across C code questions in solved questions given to us:

```
#define PBIN (volatile unsigned char *) 0xFFFFFFF3
#define PBOUT (volatile unsigned char *) 0xFFFFFFF4
#define PBDIR (volatile unsigned char *) 0xFFFFFFF5
#define PCONT (volatile unsigned char *) 0xFFFFFFF7
#define CNTM (volatile unsigned int *) 0xFFFFFFD0
#define CTCON (volatile unsigned char *) 0xFFFFFFD8
#define CTSTAT (volatile unsigned char *) 0xFFFFFFD9
#define IVECT (volatile unsigned int *) (0x20)
```

Stated in class that writing these out in full, perfect detail seemed pretty time consuming and not worth expecting of us on a timed exam. I'm likely going to write "#define [Name] ---------- 0x[Address]" to save time. Addresses can be found on the register sheets given to us.

```
interrupt void intserv();
```

Necessary to set up the ISR.

```
volatile unsigned char led = 0x4;   /* 0x0 = LED on, 0x4 = LED off */
volatile unsigned char digit = 0;   /* digit for display */
```

Sample global variables. Volatile is important when interrupts are involved.

In main:

```
*PADIR = 0xF;                       // Configure Port A.
*PAOUT = 0xF;                       // Initialize display ('0').
*PBDIR = 0x1;                       // Configure Port B.
*PBOUT = 0x1;                       // Initialize LED (Off).
```

Sample port configurations. In this case, PA was configured as an output (1111 1111) and initialized to have the connected display read 0. (1111 1111, active low).
PB was configured as 7 input bits, and 1 output bit (0000 0001), with the one output bit initialized to be off. (0000 0001, active low)

```
*IVECT = (unsigned int *) &intserv;     // Set the interrupt vector.
asm("MoveControl  PSR, #0x40");         // Respond to IRQ interrupts.
*PCONT = 0x40;                          // Enable PBIN interrupts.
    //OR
*PCONT = 0x10;                          // Enable PAIN interrupts.
    //OR
*CTCON |= 0x10;                         // Enable timer interrupts.
*CTCON &= 0xEF;                         // Disable timer interrups.
```

The first two lines set up the CPU to handle interrupts. The last set of lines are examples of enabling interrupts in specific parallel port control bits, or the timer control register.

```
*CTCON = 0x2;                       // Stop the timer (if running).
*CNTM = 100000000;                  // Setting 1-second timeout for 100Mhz Timer
*CTCON = 0x1;                       // Start the timer.
*CTSTAT = 0x0;                      // Clear the "Reached 0" flag.
while ((*CTSTAT & 0x1) == 0);       // Wait for timer to count down.
    //OR
if ((*CTSTAT & 0x1) != 0){          // Poll to see if timer has counted down.
                                    // (i.e. doesn't halt program)
```

Timer usage, with two examples of how to handle the timer counting down.

```
sample = *PAIN & 0x80;            /* Sample PAIN, isolate Bit 7 */
if (sample == 0) pressed = 1;
```

How to isolate a specific bit to check whether certain conditions have been met.

```
sample = *PBIN & 0x3;      /* Sample PBIN, isolate bits [1:0] */
if (sample == 0x1)         /* E = 0, D = 1 */
else if (sample == 0x2)    /* E = 1, D = 0 */
```

How to isolate multiple bits.

Question Type 2

Given: Priority type, table of tasks and their respective values. *(e.g.):*

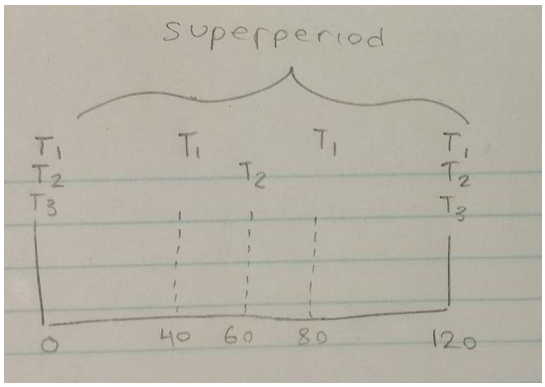
Task T_i	Period P_i	WCET C_i	Deadline D_i	Initial Delay ϕ_i
T1	40	10	40	0
T2	60	20	60	0
T3	120	40	100	0

These values represent:

- How often the task repeats on schedule. (Period)
- The duration of each task in time units. (WCET, or Worst Case Execution Time)
- The latest a task can be completed after becoming available to complete. (Deadline).
- How far past t = 0 a task becomes available. (Initial Delay)

These aspects are shared by both Rate Monotonic and Earliest Deadline First methods, as are the following steps:

1. Calculate the LCM (Least Common Multiple) given your set of task periods. We only need to determine the task scheduling for this 'superperiod', after which it will repeat.
2. Map out the times in which tasks are guaranteed to repeat. See right for an example.
3. Determine the initial priorities for each task.
    a. For Rate Monotonic, this is easy. $\tau = 1/P_i$
    b. For Earliest Deadline First, the formula $\tau = 1/(\phi_i + D_i + kP_i)$ is used instead, with k = 0 initially.
4. When there are multiple tasks available at the same point in time, the task with the greatest priority value go first. It's then drawn on the task schedule until it is completed, or until another task becomes available with greater priority. *(e.g., the repeating tasks noted in step 2.)*
    a. For Rate Monotonic, the priority value remains the same for every instance of a task.
    b. For Earliest Deadline First, the k value will increase and so a new priority value must be calculated, and new comparisons must be made when two tasks are both available.
5. Repeat with the next highest priority task until each task has been completed, which should be before the end of the superperiod.



Notes:

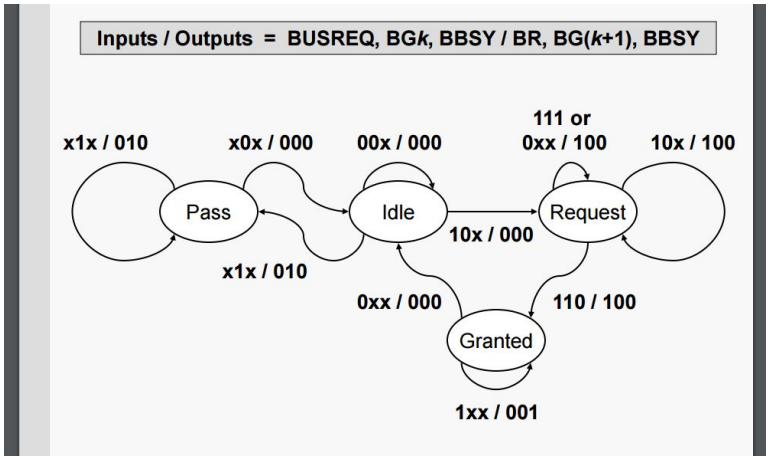
- If two tasks are tied with equal priority values, which one goes first can be chosen arbitrarily.
- Do a double check as to whether each task has been completed before its deadline. I don't think that a situation will be presented in which we have to account for this, but it's possible that one arbitrarily chosen tiebreaker causes deadline conflicts.

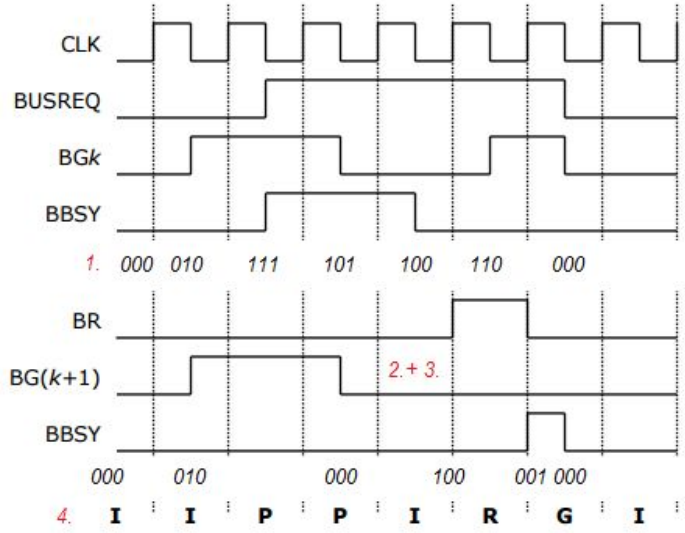
Question Type 3

a. FSM State Diagram Waveforms

Given: State diagram, input timing diagram, starting state.

Needed: Output timing diagram.





1. At each change in the input waveform, write down the entire input in binary form. This makes determining which path to take when leaving a state a lot easier.

2. Starting from the given initial state, compare the t = 0 input with the possible input conditions for leaving that state.
   Follow the path corresponding to your input bit values, and draw the output bit values on your waveform diagram at the same rising/falling edge of the clock. his is like step 1, but backwards.
   A mealy state diagram will display its output on the path, while a moore state diagram will display its output on the new state.

3. Repeat this step for every rising and falling edge of the clock. Note that often times the input bits will change, but there will be no change in states. This is because of ‘x’ in bit values representing 1 or 0. *(i.e. 11x is valid for 111 or 110)*

4. Label each clock cycle with the state that the system was in during that clock cycle. A state label starts at each rising edge of the clock cycle.

b. Daisy Chain Arbitration Scheme Waveforms

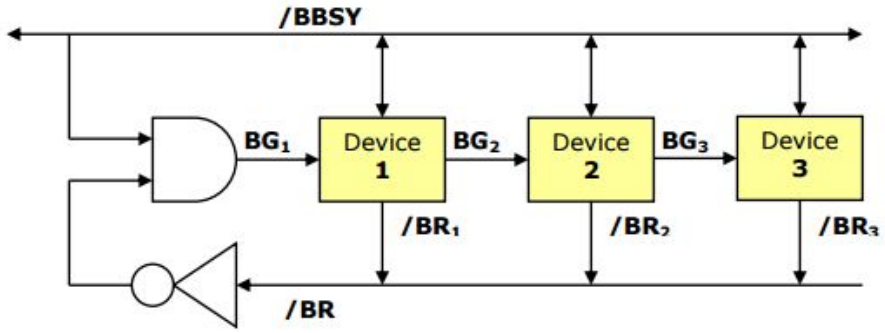
Given: Daisy chain arbitration scheme, incomplete timing diagram for scheme, assumptions for device functionality, time delay information.

Needed: Completed timing diagram.

1. Using the given arbitration scheme, and the assumptions and information in the question, make a set of rules that show how changes in some bits cause changes in others. Also note the delay in the propagation of changes.

   I'll use Assignment 3, Q4 as an example.

4. [5 points] Consider the daisy-chain arbitration scheme shown below. Assume that the input-to-output signal propagation delays are the same and equal to **d** for all three devices, the inverter, and the **AND** gate. Also, assume that device **x** is able to start using the bus (making **/BRx = 1** and **/BBSY = 0**) only when it receives a **0-1** transition on its bus-grant input **BGx** and detects that the bus is not currently busy (i.e., **/BBSY = 1**). Also, assume that device **x** lets the bus-grant propagate through only when it is neither requesting nor using the bus. Finally, assume that any of the three devices will need to use the granted bus for only **3d** time units. Complete the timing diagram shown below, where Device 1 and Device 3 request the bus at the same time **t = 0**.



Cause	Effect	Comments	Delay
Δ /BBSY	Δ BG1	Just AND gate.	d
Δ /BR	Δ BG1	AND gate, *and* inverter.	2d
Δ BGx (0 -> 1) and /BRx = 0	Δ /BRx (0 -> 1) Δ /BBSY (1 -> 0)	Device is ready to start using bus,	d
Δ /BBSY (1 -> 0)	Δ /BBSY (0 -> 1)	Bus usage duration.	3d
Δ BGx	Δ BG(x+1)	Only when BGx is not accessing or requesting the bus.	d

2. Using these rules to complete the question looks like this:
   i. BR (1 -> 0) causes BG1 to shift from (0 -> 1) with a delay of 2d.
   ii. A shift in BG1 (0 -> 1) causes /BR1 to shift from (0 -> 1), and /BBSY to shift from (1 -> 0), with a delay of d.
   iii. A shift in /BBSY (1 -> 0) causes a shift in BG1 (1 -> 0) with a delay of d.
   iv. A shift in /BBSY (1 -> 0) also causes a shift in /BBSY (0 -> 1), with a delay of 3d.
   v. A shift in /BBSY (0 -> 1) causes a shift in BG1 (0 -> 1), and, now that device 1 has stopped accessing or requesting the bus, that shift can cause a shift in BG2 (0 -> 1) with a delay of d. Because device 2 is not requesting or accessing either, this change shifts to BG3 (0 -> 1) with a delay of d.
   vi. A shift in BG3 (0 -> 1) causes /BR3 to shift from (0 -> 1) and /BBSY to shift from (1 -> 0), with a delay of d.
   vii. A shift in /BBSY (1 -> 0) causes a shift in BG1 (1 -> 0). Device 1 is not accessing or requesting the bus, so that shift causes a shift in BG2 (1 -> 0) with a delay of d. Because device 2 is not requesting or accessing either, this change shifts to BG3 (1 -> 0) with a delay of d.
   viii. A shift in /BBSY (1 -> 0) also causes a shift in /BBSY (0 -> 1) with a delay of 3d.
   ix. No devices are requesting the bus. Problem is complete.