(20) **Question 1.** For each situation, choose the best interface protocol to implement a serial channel. In particular, for each part simply write in your selection from the available choices (i.e., digital, RS232, RS422, or RS485). You may use the same protocol more than once, or not at all.

<table>
<thead>
<tr>
<th>Baud rate</th>
<th>Distance</th>
<th>Duplex</th>
<th>Noise</th>
<th>Your choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>30 ft</td>
<td>half duplex</td>
<td>very high</td>
<td></td>
</tr>
<tr>
<td>9600</td>
<td>30 ft</td>
<td>full duplex</td>
<td>very high</td>
<td></td>
</tr>
<tr>
<td>9600</td>
<td>3000 ft</td>
<td>full duplex</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>500,000</td>
<td>30 ft</td>
<td>full duplex</td>
<td>medium</td>
<td></td>
</tr>
<tr>
<td>500,000</td>
<td>3 in</td>
<td>full duplex</td>
<td>low</td>
<td></td>
</tr>
</tbody>
</table>
**Question 2.** The overall objective of this problem is to convert this gadfly SCI example to generate and receive even parity frames. Redesign the system so that even parity is generated on each 11-bit transmit frame that is output (1 start, 8 data, even parity, and 1 stop). You should also check for parity errors on each 11-bit receive frame that is input. There is a protected global variable, which you will initialize to zero, and increment on every input frame that is received in error.

```c
unsigned short ParErr; // number of parity errors detected by InChar()
```

The following are the ritual, input, and output functions from SCI2.C. Please make the necessary changes.

```c
void InitSCI(void){
    SC0BDH=0;
    SC0BDL=13;
    SC0CR1=0;
    SC0CR2=0x0C;
}

char InChar(void){
    while ((SC0SR1 & RDRF) == 0){};
    return(SC0DRL);
}

void OutChar(char data){
    while ((SC0SR1 & TDRE) == 0){};
    SC0DRL = data;
    if(data==CR){ /* if CR add LF */
        while((SC0SR1 & TDRE) == 0){};
        SC0DRL = LF;)
    else if(data==LF){ /* if LF add CR */
        while((SC0SR1 & TDRE) == 0);
        SC0DRL = CR;
    }
}
```
**Question 3.** The overall objective of this problem is to record songs, which a musician plays on an 8-key piano. The keys are labeled 0 through 7. Each of the 8 piano keys is a simple switch, which is closed (0Ω) when the key is touched, and is open when the key is released. There is 100 µs of bounce. You will design the hardware interface to the piano and write the song-recording software. The main program, which you do not write, will call your ritual then perform unrelated operations. I.e., the song recording will occur in the background using interrupt synchronization. Time will be expressed as a 16-bit unsigned decimal fixed-point number, with a resolution of 0.01 seconds. Time will vary from 0 to 655.35 seconds. Your system will measure the time when a key is pressed, and when it is released. The data structure used to store the song is as follows:

```c
struct Event {
    unsigned char KeyCode; // key number, 0 through 7
    unsigned char What;   // 0 means touch, 1 means release
    unsigned short When;  // Time in 0.01 sec units
}
```

```c
typedef struct Event EventType;
EventType Song[200]; // a song consists of up to 200 events
```

If the musician touches keys 1,3,5 at time 1.2345 sec, and releases them all at 9.8765 sec, the six entries would be:

- `Song[0]={1,0,124}` means key 1 is touched at 1.24 sec
- `Song[1]={3,0,124}` means key 3 is touched at 1.24 sec
- `Song[2]={5,0,124}` means key 5 is touched at 1.24 sec
- `Song[3]={1,1,987}` means key 1 is released at 9.88 sec
- `Song[4]={3,1,987}` means key 3 is released at 9.88 sec
- `Song[5]={5,1,987}` means key 5 is released at 9.88 sec

(10) **Part a)** All eight keys will be interfaced to a single I/O port. Which port would you use? Justify?

(10) **Part b)** Show the hardware interface between the piano and the 6812. Since each of the piano keys, numbered 0 to 7, is separate and identical, you need only show the connection between Key0 and bit 0 of the port you selected in part a. Decide whether to debounce in hardware or software. Select the best approach (direct, scanned, or multiplexed.)

(5) **Part c)** Show any additional global data structures required.
(15) Part d) Show the ritual that initializes the I/O port and data structures. You are free to use any of the available interrupt mechanisms. Arm and enable interrupts.

(15) Part e) Show the ISR(s), which record the song. Stop recording (disarm) after 200 events are recorded or after 655.35 seconds, whichever occurs first.

(5) Part f) Show the C code that establishes the necessary interrupt vector(s).