(20) **Question 1.** Count the number of falling edges of an input signal connected to PH2 using interrupts.

Part a) Show the C code to initialize Port H bit 2 (PH2) to interrupt on the fall of this input signal.

```c
#define PH2 0x04
void Init(void){
    asm(" sei");       // make ritual atomic (optional)
    DDRH &= ~PH2;   // PH2 is an input (friendly)
    KWIEH |= PH2;    // arm PH2 (friendly)
    KWIFH = PH2;     // clear flag (optional step)
    counter=0;       // initialize data
    asm(" cli");       // enable interrupts
}
```

Part b) Show the ISR that increments the `counter` on every fall of PH2.

```c
#pragma interrupt_handler KeyWakupHhandler()
void KeyWakupHhandler(void){
    KWIFH = PH2;  // ack, clear flag
    counter++;
}
#pragma abs_address:0xffce
void (*the_vector[][]) = { KeyWakupHhandler}
```

Part c) The *read-modify-write* sequence in the above ISR is not a critical section because interrupts are disabled.

(15) **Question 2.** Interface a door lock/unlock solenoid to the microcomputer. The current is 5V/50Ω=100mA.

Part a) The 1N4003 diodes are used to eliminate back EMF. The ULN2074 in open-emitter mode are used to source the 100 mA current. PNP transistors like the 2N2907, TIP30A, TIP32A, TIP42A, or TIP125 could have been used.

```
PH1 +5V
   b c
  1N4003
  PH0 +5V
   b c
  1N4003
     ULN2074
+5V
   b c
ULN2074
   1N4003
       +5V
   b c
  ULN2074
   +5V
   b c
ULN2074
   +5V
   b c
  ULN2074

PH1 +5V
   b c
  1N4003
  PH0 +5V
   b c
  1N4003
       PNP
          +10kΩ

+5V
   b c
   PNP
ULN2074
   +5V
   b c
   PNP
ULN2074
   +5V
   b c
   PNP
ULN2074
   +5V
   b c
   PNP
ULN2074
```

Part b) Show the ritual that initializes the interface.

```c
#define PH1 0x02  // lock coil
#define PH0 0x01  // unlock coil
void Init(void){
    DDRH |= PH1+PH0;     // PH1 PH0 are outputs (friendly)
    TSCR = 0x80;        // enable timer
    PORTH &= ~(PH1+PH0);  // disable both solenoids (friendly)
}
```

Part c) A `lock()` function that energizes the lock solenoid for exactly 1 second.

```c
void wait1Sec(void){ int j; short EndT; // TCNT at the end of the delay
    EndT = TCNT;
    for(j = 0; j < 1000; j++){
        if(EndT & (short)TCNT>0){ // 1 ms inner loop
            while(EndT-(short)TCNT>0);  // wait until TCNT passes EndT
        }
    }
    PORTH |= PH1;   // PH1=1 activate the lock solenoid (friendly)
    wait1Sec();
    PORTH &= ~PH1;  // PH1=0 deactivate the lock solenoid (friendly)
}
```

Part d) An `unlock()` function that energizes the unlock solenoid for exactly 1 second.

```c
void unlock(void)
    PORTH | = PH0;  // PH0=1 activate the unlock solenoid (friendly)
    wait1Sec();
    PORTH &= ~PH0;  // PH0=0 deactivate the unlock solenoid (friendly)
```
**Question 3.** The SPI port of one 6812 is connected to the SPI port of another 6812.

**Part a)** Since the two shift registers are exchanged: 3) data can only flow in both directions.

**Part b)** Give the initialization values in hexadecimal for each SPI.

<table>
<thead>
<tr>
<th></th>
<th>Master</th>
<th>Slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDRS</td>
<td>0xE0</td>
<td>0x40</td>
</tr>
<tr>
<td>SP0CR1</td>
<td>0x52</td>
<td>0x42</td>
</tr>
<tr>
<td>SP0CR2</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>SP0BR</td>
<td>0x00</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Other answers for SP0CR1 are OK as long as bits 7,3,2,0 are the same in both.

**Question 4.** An expanded mode 6812 is initialized to have 3 stretches (the total cycle time is 500 ns).

**Part a)** \( RDR = \text{Read Data Required} = (500 - 30, 500 + 0) = (470, 500) \)

**Part b)** \( WDA = \text{Write Data Available} = (60 + 46, 500 + 20) = (106, 520) \)

**Question 5.** The I bit is automatically set by the hardware after the registers are pushed on the stack, but before the ISR executes. The I bit is cleared by the \texttt{rti} instruction when the CCR is pulled from the stack.

**Question 6.** Almost identical to the example in Chapter 4.

```c
void InitFifo(void){ char SaveSP;
    asm("tpa\n    staa %SaveSP\n    sei"); /* make atomic, entering critical*/
    PutI=GetI=Size=0; /* Empty when Size is 0 */
    asm("lda %SaveSP\n    mp"); /* end critical section */
}

int PutFifo(short data){ char SaveSP;
    if(Size == 256 ){
        return(0); /* Failed, fifo was full */
    }
    else{
        asm("tpa\n        staa %SaveSP\n        sei"); /* make atomic, entering critical*/
        Size++;
        Fifo[PutI] = data; /* put data into fifo */
        PutI++; /* automatically wraps */
        asm("lda %SaveSP\n        mp"); /* end critical section */
        return(-1); /* Successful */
    }
}

int GetFifo(short *datapt){ char SaveSP;
    if(Size == 0 ){
        return(0); /* Empty if Size=0 */
    }
    else{
        asm("tpa\n        staa %SaveSP\n        sei"); /* make atomic, entering critical*/
        *datapt = Fifo[GetI];
        Size--;
        GetI++; /* automatically wraps */
        asm("lda %SaveSP\n        mp"); /* end critical section */
        return(-1);
    }
}
```

**Question 7.** A 7405, 74S05, 74LS05 or 7406 can sink the 8 mA current. The \( R=\frac{5-2.1-0.5}{8\text{mA}}=300\Omega \).

**Question 8.** Acknowledgement is clearing the flag that requested the interrupt.

**Part a)** You can clear RDRF by reading the status register with RDRF set, followed by reading the SCI data register.

**Part b)** You can clear TDRE by reading the status register with TDRE set, followed by writing the SCI data register.

**Part c)** You can clear C7F by writing a one to its bit location (bit 7) in the TFLG1 register.

**Part d)** You can clear TOF by writing a one to its bit location (bit 7) in the TFLG2 register.

**Part e)** You can clear RTIF by writing a one to its bit location (bit 7) in the RTIFLG register.