(5) Question 1. Q) I/O bound is the condition when the bandwidth of the system, which is the amount of data processed per second, is limited by the speed of the input device.

(5) Question 2. K) vectored interrupt is an interrupt scheme where each individual flag that can request an interrupt has a unique interrupt vector.

(5) Question 3. Y) intrusive is a condition when the debugger itself significantly alters the operation of software/hardware system.

(5) Question 4. P) stabilization is a debugging technique that fixes all its inputs to specific values and can be repeated over and over.

(5) Question 5. V) tristate is a type of logic that can be high, low or off.

(5) Question 6. A) latency is the time between new input being ready and the time when the new input is read.

(5) Question 7. I) buffered I/O is an I/O interfacing technique that uses a FIFO queue to pass data between the foreground and the background.

(5) Question 8. There are two possible causes of the FIFO full condition. The FIFO will become full if the average producer rate exceeds the average consumer rate. In this case, increasing the FIFO size will not correct the error. In this case, one must increase the consumer rate (or decrease the producer rate). The other condition that can cause the FIFO to become full is if there is a temporary condition where the producer rate (r_P) exceeds the consumer rate (r_G). The excess number of elements in the FIFO will be $n=\int (r_P-r_G)dt$. If n>FIFO size then it will get full. In this case we can solve the problem either by increasing the FIFO size or by increasing the consumer rate or by decreasing the producer rate.

D) Sometimes the problem can be solved by increasing the size of the FIFO, but other times it will be necessary to increase the consumer rate or decrease the producer rate.

(5) Question 9. value = integer*resolution = 41/4 = **10.25**

(5) Question 10. value = integer*resolution. The range of 16-bit signed integers is -32768 to +32767. The smallest resolution is 0.01, which would create a value range of -327.68 to +327.67. Many students confused this question. A decimal fixed-point scheme has a resolution that is a power of 10. E.g., …1000, 100, 10, 1, 0.1, 0.01, 0.001 …

(5) Question 11. When active the output voltage of PT0 will be V_{OL}, and the LED voltage will be V_D. This means $(5 - V_{OL} - V_D)$ will develop across the resistor. To generate the desired LED current of I_D, the resistor should be

$$R = (5 - V_{OL} - V_D)/ I_D$$

(5) Question 12. There are two possible sequences

I) **(optional)** The periodic timer clock times out, setting `RTIF`
D) The CC, A, B, X, Y, PC registers are pushed on the stack
E) The `I` bit is set to one (disable)
A) The PC is loaded with the 16-bit contents of `$FFF0`

or **I)** **(optional)** The periodic timer clock times out, setting `RTIF`
D) The CC, A, B, X, Y, PC registers are pushed on the stack
A) The PC is loaded with the 16-bit contents of `$FFF0`
E) The `I` bit is set to one (disable)

(5) Question 13. value = integer*resolution. The range of integers will be –50*256 to +50*256, which is –12800 to +12800. These integers will fit into

B) `short`, which has a range of -32768 to +32767

(35) Question 14.

```
int Left2Find;             // switch pushes
unsigned char LastSwitch; // previous  ......  H
// =0x80  was not pressed last time, =0x00  was pressed last time
// initialize RTI every 16.384ms
void RTIinit(void){
  asm sei          // Make atomic
  DDRAD &= ~0x80;  // PA7 input switch  ...... A
  ATDDIEN |= 0x80; // enable AD digital ...... B
  DDRT |= 0x01;    // PT0 output to LED ...... C
  PTT &= ~0x01;    // light off         ...... D
  CRGINT = 0x80;   // RTIE=1 enable rti ...... E
  RTICTL = 0x71;   // 125ns*65536*2     ...... F
  Left2Find = 5;   // looking for 5
  LastSwitch = PTAD&0x80; // 0=pressed
  asm cli                               ...... G
}
void interrupt 7 RTIhandler(){          ...... I
unsigned char thisTime;
  CRGFLG = 0x80;       // acknowledge,  ...... J
  thisTime = PTAD&0x80;                 ...... K
  if(LastSwitch==0){ // last pushed?
    if(thisTime){    // now released?
      Left2Find--;                      ...... L
      if(Left2Find==0){  // done?
        PTT |= 0x01; // activate LED    ...... M
        CRGINT = 0x80;   // disarm
      }
    }
  }
  LastSwitch = thisTime;                ...... N
}
```

Grading codes (penalties)

A(-3) You need to make PA7 an input by clearing the corresponding bit in the direction register. Typically this is done once in the ritual, and not inside the ISR. Friendly software just clears bit 7.

B(-3) You need to make PA7 a digital signal by setting the corresponding bit in `ATDDIEN`. Typically it is a good idea to set `DDRAD` first, before enabling `ATDDIEN`. Friendly software just sets bit 7.

C(-3) You need to make PT7 an output by setting the corresponding bit in the direction register. Typically this is done once in the ritual, and not inside the ISR. Friendly software just sets bit 0.

D(-3) You need to initially make the LED off. Coming up out of a power on reset, you might be tempted to skip this step, thinking that the default output value will be zero. It is good design to explicitly set important initial values, so that the system can be reinitialized in software, without having to hit the reset, or remove the battery. Friendly software just clears bit 0.

E(-3) RTI is armed by setting bit 7. Since we are using all of the RTI (we are not sharing it with other software modules), issue of friendly does not apply to `CRGINT`.

F(-3) The RTI period must be longer than the switch bounce, so that there will be at most one interrupt during the bouncing. Since we are using all of the RTI (we are not sharing it with other software modules), issue of friendly does not apply to `RTICTL`.

G(-3) Interrupts need to be enabled. Typically this is done last, after everything is initialized. When there are many interrupt sources, it might be a good idea to initialize all the devices, then enable interrupts.

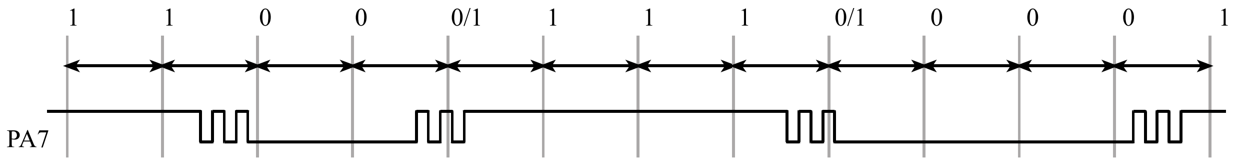
H(-3) There must be permanent storage in the problem, which are shared between the ritual and the ISR. If you define the variables inside the ritual or inside the ISR (even if you make them `static`), only one module can access them.

I(-3) I was specifically looking for you to know the Metrowerks syntax to define a ISR. This code also defines the interrupt vector.

J(-3) Every ISR must either acknowledge (clear flag that triggered the interrupt) or disarm. Since we are interested in additional interrupts, we must clear the flag. Since we are using all of the RTI (we are not sharing it with other software modules), issue of friendly does not apply to `RTIFLG`. I didn't take off for the following poor code

```
  CRGFLG |= 0x80;
```

but remember this will be a bug when using a flag register like TFLG1 with multiple flags.
K(-3) To observe one particular bit, our software reads the port and masks away the unwanted bits. Read cycles to most I/O ports do not change them, so it will be friendly.
L(-3) The software should count the number of pushes.
M(-3) You need to turn on the LED. Friendly software just sets bit 0. If you exclusive or bit 0, then the LED will go off after pushing it 65541 times.
N(-3) You need a way to detect the switch was released.
P(-5) Backward jumps that wait for events (which are large compared to 16ms) are not appropriate. The key to solving this problem is to consider what the signal looks like when sampled every 16 ms. You are guaranteed at most one interrupt during the bounce period, so it doesn't matter if the input read during a bounce is high or low. You are guaranteed at least one high during the release and at least one low during the push.



Q(-3) Many solutions had this bug: if the button were pushed and held down, then multiple counts were recorded.
R(-3) It is inappropriate to include `asm sei` at the beginning and `asm cli` at the end of an ISR, because the hardware automatically sets I=1 during the context switch, and the `rti` instruction at the end of the ISR will restore I back to 0 at the end of the ISR.