(5) Question 1. 10k in parallel with 10k is 5k, using the rule R1 || R2 = (R1*R2)/(R1+R2). The total resistance is 15k. Using the voltage divider equation V = 3.3V(5k/15k) = 1.1V. Another solution first uses Ohm's Law to calculate current I = 3.3V/15k, then uses Ohm's Law again,

V = 5k*I = 3.3V(5k/15k) = 1.1V.

(5) Question 2.
(3) Part a) TCNT is running at 125ns times 4, which is 500ns. The output compare 7 interrupt occurs every 100 TCNT cycles, which is 50 μsec.
(2) Part b) The sampling rate is determined by the interrupt frequency, 1/50μs is 20 kHz. According to the Nyquist Theorem, the largest frequency component faithfully represented in the data in the buffer will be 10 kHz (one half the sampling rate.)

(10) Question 3. The trick is you have to wait for both the rising and falling edges.
```
void main(void){ unsigned char count=0;
  DDRT |= 0x01;  // PT0 is output
  DDRT &= ~0x02; // PT1 is input
  while(1){
    while((PTT&0x02)==0){}; // wait until rising edge
                            // PT1 is now high
    count++;
    if((count&0x01)==0){    // count is even
      PTT |= 0x01;          // pulse
      PTT &= ~0x01;
    }
    while(PTT&0x02){};      // wait until falling edge
                            // PT1 is now low
}
```

(20) Question 4. We need a shared global pointer. Clear TDRE by read status, write data
```
unsigned char *Pt;
void SCI1_Output(unsigned char *Buffer){
  if(Buffer[0] == 0) return;   // ignore empty buffers
  SCI1CR1 = 0;
  SCI1CR2 = 0x88;     // or 8C, TIE arm and TE enable
  SCI1BD = 8000/16/5; // 8MHz/16/5kHz =100
  Pt = Buffer;
asm cli
}
void interrupt 21 SCI1Handler(void){ // TDRE trigger
  if((*Pt) == 0){       // disarm after last character sent
    SCI1CR2 = 0x08;     // or 0x0C, TIE disarm and TE enable
  } else{
    if(SCI1SR1&0x80){   // read status with TDRE = 1
      SCI1DRL = (*Pt);  // write data (acknowledge TDRE)
      Pt++;
    }
  }
}
```

(6) Question 5.
(2) Part a) Flow rate is -10 L/min is 25% between min and max, so ADC will be 4096*25% = 1024
(2) Part b) The resolution allowed by the ADC will be 40 L/min = 4096, which is about 40/4000 = 10/1000 = 1/100 = 0.01 L/min. I would use a decimal fixed-point resolution of 0.01 L/min.
(2) Part c) Since the ADC is 12 bits, I would use 16-bit precision for the fixed-point number system.

(4) Question 6. For the C bit, first convert to unsigned, -1 means 255. So 1+255 will cause an unsigned overflow, setting the C bit to 1. The result in Register A will be 0, so the Z bit will be 1.

(5) Question 7. First fetch the four bytes the machine code, then read from PTT, and lastly write to PTT.

R/W	Addr	Data
R	$4065	$1C
R	$4066	$02
R	$4067	$40
R	$4068	$01
R	$0240	$08
W	$0240	$09

(5) Question 8. For each application choose the term that *best* matches.

Application	**Debugging term**
Measuring where and when software executes	Profile
Can be used record data during execution without pausing	Scanpoint (similar to a dump)
Debugging with a small but inconsequential effect on the system itself	Minimally intrusive
Adding a LCD to display important variables during execution; the LCD is not part of the necessary components of the system.	Monitor or highly intrusive
Flashing an LED letting the user know the software is running	Heartbeat

(15) Question 9. Design and implement a FIFO that can hold up to 4 elements. Each element is 3 bytes. There will be three subroutines: `Initialization`, `Put` one element into FIFO and `Get` one element from the FIFO.

(4) Part a) Show the RAM-based variables are available, and NO additional storage may be allocated

```
Fifo  rmb 5*3  ; room for 4 elements
PutPt rmb 2    ;place to put
GetPt rmb 2    ;place to get
```

(4) Part b) Write an assembly function that initializes the FIFO.

```
Init ldx  #Fifo
     stx  PutPt
     sty  GetPt
     rts
```

(4) Part c) Write an assembly function that puts one 3-byte element into the FIFO.

```
Put  ldd  #0
     ldx  PutPt
     movw 0,y,2,x+  ;copy three bytes
     movb 2,y,1,x+
     cpx  #Fifo+15  ;need to wrap
     bne  Pok
     ldx  #Fifo
Pok  cpx  GetPt     ;check for full
     beq  Pout      ;skip if full
     stx  PutPt     ;data stored ok
     ldd  #1        ;success
Pout rts
```

(3) Part d) Write an assembly function that gets one 3-byte element from the FIFO.

```
Get  ldd  #0
```

```
     ldx  GetPt
     cpx  PutPt
     beq  Gout      ;skip if empty
     movw 2,x+,0,y  ;copy three bytes
     movb 1,x+,2,y
     cpx  #Fifo+15  ;need to wrap
     bne  Gok
     ldx  #Fifo
Gok  stx  GetPt     ;data retrieved ok
     ldd  #1        ;success
Gout rts
```

(5) Question 10. The state sequence will be Stop,Go,Turn,Go,Turn… switching back and forth between Go and Turn. The sequence of outputs will be 7,3,5,3,5,3,5,3,5,…

(10) Question 11. Consider output compare 7 interrupts.

(3) Part a) The three events are

Arm, C7I in TIE must be set by software

Enable, I=0 in CCR must be cleared by software, via the cli instruction

Trigger, C7F in TFLG1 must be set by hardware, when TCNT equals TC7

(4) Part b) The events that occur as the computer switches from foreground to background are

Finish instruction (can skip this step for full credit on the question)

Push PC,Y,X,A,B,CCR on stack

Set I=1 (to prevent interrupt from interrupting itself)

Set PC = vector address at $FFE0, which will be **`TC7Handler`**

(3) Part c) Write assembly code to acknowledge an output compare 7 interrupt.

```
  ldaa  #$80
  staa  TFLG1
```

or

```
  movb  #$80,TFLG1
```

(10) Question 12. In this question, you will translate the C code into 9S12 assembly.

C code	Assembly

`void main(void){`
`unsigned short c;`
`unsigned char d;`
`  c = 0;`
`  for(;;){`
`    d = PTT;`
`    if(d&0x01){`
`      c = c+d;`
`    }`
`  }`
`}`
	`main lds  #$4000`
	`     leas -3,sp  ;allocate c,d`
	`     tsy`
	`c    set  0`
	`d    set  2`
	`     movw #0,c,Y`
	`loop ldab PTT`
	`     stab d,y`
	`     bitb #1`
	`     beq  skip`
	`     clra       ;RegD = d`
	`     addd c,y   ;RegD = c+d`
	`     std  c,y   ;c = c+d`
	`skip bra  loop`
	`     org  $FFFE`
	`     fdb  main`