Jonathan W. Valvano
April 7, 2000, 11:00am-11:50am

(20) Question 1. We usually try to divide last. `x/100` returns 0, 1, or 2. So `(x/100)*49` will be 0, 49, or 98.

```
   y=(49*x+51*y)/100; // compiler will use 16-bit intermediate values
```

(20) Question 2. There is a potential overflow. If x is greater than 1337, then `49*x` will be greater than 65535. There is a similar problem with `51*y`. If y is greater than 1285, then `51*y` will be greater than 65535. The solution is to use 32-bit intermediate calculations. In C, we cast the operations to `long`.

```
   y=(49*(long)x+51*(long)y)/100;}
```

The trouble with this is speed. All calculations take 32-bit inputs and generate 32-bit results. In assembly, we can perform a 16-bit by 16-bit multiply that generates a 32 bit product. The additions will be full 32-bit. But, the divide will be 32 bit dividend by 16-bit divisor yielding a 16 bit quotient. See the **emacs** and **edivs** instructions. This assembly implementation is *much* faster than the compiled version.

```
xn   ds    2      ; 12-bit raw data
yn   ds    2      ; 12-bit filter output
coef .word 49,51  ; filter coeficients (16-bit)
sum  ds    4      ; intermediate sum=49*x+51*y (32-bit)
TC5handler
     movb  #$20,TFLG1 ; ack
     ldd   TC5
     addd  #8333
     std   TC5        ; fs=240 Hz
     movw  #0,sum     ; sum=0 (32-bit)
     movw  #0,sum+2
     ldx   #xn        ; pointer to xn,yn
     ldy   #coef      ; pointer to 49,51
     emacs sum        ; sum = 49*xn
     emacs sum        ; sum = 49*xn + 51*yn
     ldy   sum        ; Y:D = 49*xn + 51*yn
     ldd   sum+2
     ldx   #100
     edivs            ; Y=(49*xn+51*yn)/100
     sty   yn
     rti
```

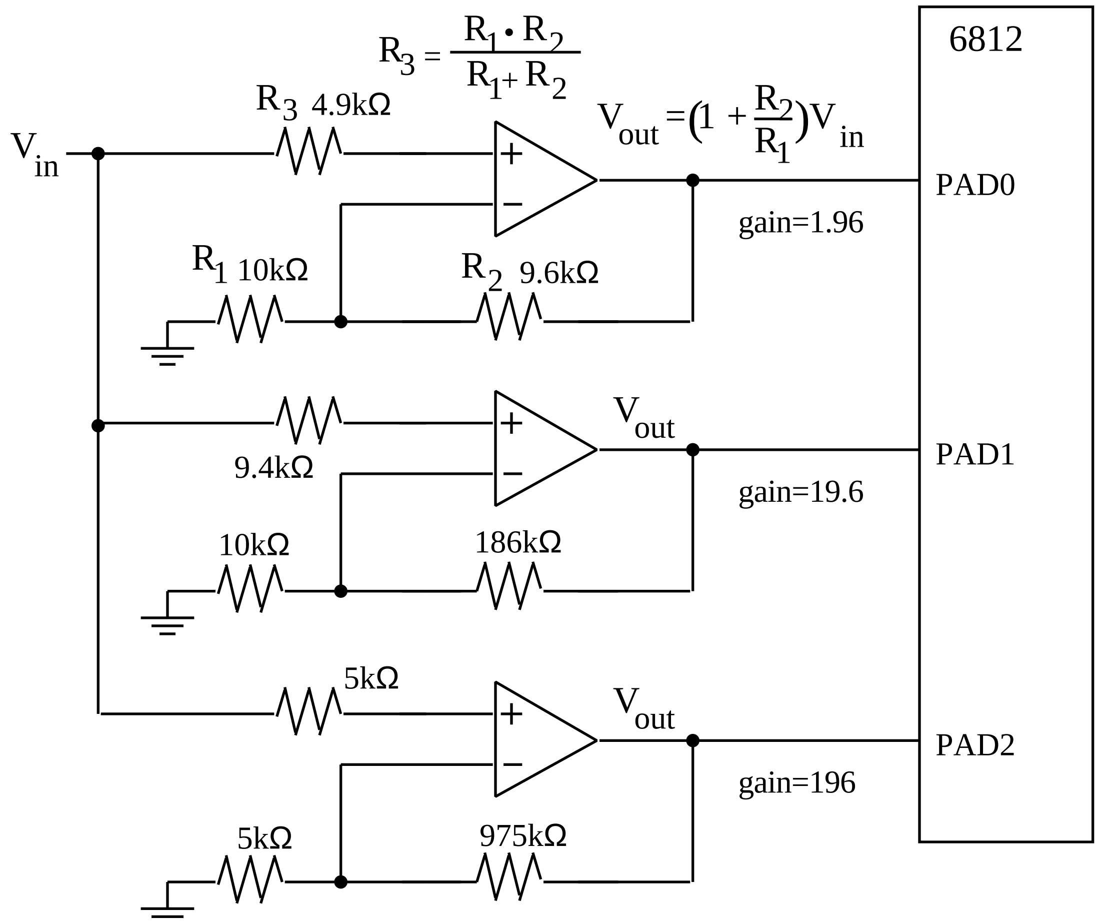
(20) Question 3. You can use the simple TCNT method to measure execution speed

```
unsigned short errorcount; // number of lost data points
#pragma interrupt_handler TC5handler()
void TC5handler(void){ unsigned short start,delay;
   TFLG1=0x20;      // ack C5F
   TC5=TC5+1000;    // fs=1000Hz
   start=TCNT;
   ProcessDAS();
   delay=TCNT-start;
   if((delay>1000)&&(errorcount<65535U))errorcount++
}
void ritual(void) {
asm(" sei");     // make atomic
  errorcount=0;
  TIOS|=0x20;    // enable OC5
  TSCR|=0x80;    // enable
  TMSK2=0x33;    // 1 us clock
  TMSK1|=0x20;   // Arm output compare 5
  TFLG1=0x20;    // Initially clear C5F
  TC5=TCNT+1000;
asm(" cli"); }
```

(40) Question 4. Select the gain to match the 5 V range of the ADC

minimum voltage (mV)	maximum voltage (mV)	resolution (mV)	precision	desired gain
0	25.5	0.1	256	196
25.5	255	1	230	19.6
255	2550	10	230	1.96

(20) Part a) All three circuits use noninverting gain. The TLC2274 chip has 4 op amps and is powered by +5, and all resistors are 1% metal film.



(10) Part b) The ritual simply enables the ADC port.

```
void InitializeDVM(void){
    ATDCTL2 = 0x80;}   // activate ADC
```

(10) Part c) The function uses MULT mode to sequentially measure all three channels. If the result is $FF, then the gain of that channel is too high.

```
void MeasureDVM(void){
   ATDCTL5= 0x10;   // start a scanned sequence of 4 samples on channels 0,1,2,3
   while((ATDSTAT&0x8000)==0){};  // wait for completion
   if(ADR2<255){
      voltage = ADR2;
      exponent = -1;}  // range 0 to 25mV
   else if(ADR1<255){
      voltage = ADR1;
      exponent = 0;}  // range 25 to 255mV
   else {
      voltage = ADR0;
      exponent = 1;}  // range 255 to 2550mV
)
```