**1.)**

**2.)**

**a.)**

Code:

asm volatile("nop");

1ce: 00 00 nop

result\_8 = a-b;

1d0: 18 1b **sub** r17, r24

1d2: 10 93 0a 01 **sts** 0x010A, r17 ; 0x80010a <result\_8>

asm volatile("nop");

Ans: 1+2 = 3 Clock Cycles

Code:

asm volatile("nop");

1e2: 00 00 nop

result\_16 = aa-bb;

1e4: 08 1b **sub** r16, r24

1e6: 19 0b **sbc** r17, r25

1e8: 10 93 09 01 **sts** 0x0109, r17 ; 0x800109 <result\_16+0x1>

1ec: 00 93 08 01 **sts** 0x0108, r16 ; 0x800108 <result\_16>

asm volatile("nop");

Ans: 1+1+2+2=4 Clock Cycles

Code:

asm volatile("nop");

result\_32 = aaa-bbb;

210: 84 1b **sub** r24, r20

212: 95 0b **sbc** r25, r21

214: a6 0b **sbc** r26, r22

216: b7 0b **sbc** r27, r23

218: 80 93 04 01 **sts** 0x0104, r24 ; 0x800104 <\_\_data\_end>

21c: 90 93 05 01 **sts** 0x0105, r25 ; 0x800105 <\_\_data\_end+0x1>

220: a0 93 06 01 **sts** 0x0106, r26 ; 0x800106 <\_\_data\_end+0x2>

224: b0 93 07 01 **sts** 0x0107, r27 ; 0x800107 <\_\_data\_end+0x3>

asm volatile("nop");

Ans: 1+1+1+1+2+2+2+2=12 Clock Cycles

**b.)**

For 8 bits it takes 3/16 = 0.1875microseconds

For 16 bits it takes 4/16 = 0.25microseconds

For 32 bits it takes 12/16 = 0.75microseconds

**c.)**

Code:

asm volatile("nop");

1d0: 00 00 nop

result\_8 = a/b;

1d2: 8f 2d **mov** r24, r15

1d4: 0e 94 dc 01 **call** 0x3b8 ; 0x3b8 <\_\_udivmodqi4>

1d8: 80 93 0a 01 **sts** 0x010A, r24 ; 0x80010a <result\_8>

asm volatile("nop");

Ans: 1+4+2=7 Clock Cycles (Assuming Call(1) Direct Subroutine Call)

We see mov, call and sts which take in total 7/16=0.4375microseconds

Code:

asm volatile("nop");

1ea: 00 00 nop

result\_16 = aa/bb;

1ec: c7 01 **movw** r24, r14

1ee: 0e 94 e8 01 **call** 0x3d0 ; 0x3d0 <\_\_udivmodhi4>

1f2: 70 93 09 01 **sts** 0x0109, r23 ; 0x800109 <result\_16+0x1>

1f6: 60 93 08 01 **sts** 0x0108, r22 ; 0x800108 <result\_16>

asm volatile("nop");

Ans: 1+4+2+2=9 Clock Cycles (Assuming Call(1) Direct Subroutine Call)

We see movw, call and sts which takes 9/16=0.5625microseconds

Code:

asm volatile("nop");

result\_32 = aaa/bbb;

21a: 0e 94 ab 01 **call** 0x356 ; 0x356 <\_\_udivmodsi4>

21e: 20 93 04 01 **sts** 0x0104, r18 ; 0x800104 <\_\_data\_end>

222: 30 93 05 01 **sts** 0x0105, r19 ; 0x800105 <\_\_data\_end+0x1>

226: 40 93 06 01 **sts** 0x0106, r20 ; 0x800106 <\_\_data\_end+0x2>

22a: 50 93 07 01 **sts** 0x0107, r21 ; 0x800107 <\_\_data\_end+0x3>

asm volatile("nop");

Ans: 4+2+2+2+2=12 Clock Cycles (Assuming Call(1) Direct Subroutine Call)

We see call and sts which takes 12/16=0.75microseconds

**d.)**

For divide it took mov,call,mov,ldi,ldi,ldi,call,sts,sts,sts,sts so 17/16=1.0625microseconds?

For subtract it took sub,sbc,movw,ldi,ldi,call,sts,sts,sts,sts so 17/16=1.0625microseconds?

**3.)**

**a.)**

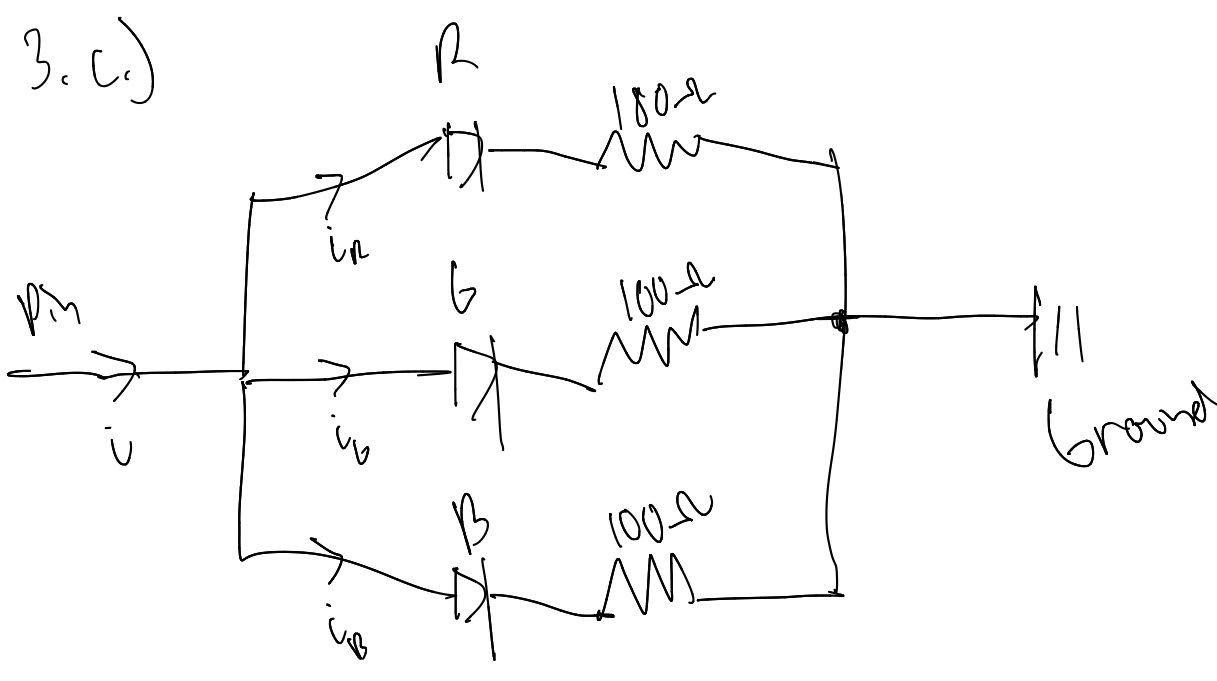
Know I\_LED=(5V-V\_LED)/R so for both green and blue it will be i=(5-3)/100=20mA

For Red LED i=(5-2.2)/180=15.56mA

**b.)**

Know DC current per I/O pin ins 40mA so the blue and green could have 2 in series and the red could be 2 or if we increase the voltage source to 6, we could fit 3 red leds

**c.)**



**4.)**

**a.)**

#define button (1<<2) //PC2 is A2 in analog in

DDRC &= ~button //to set as input

PORTC |= button // set the output to enable pullup resistor

**5.)**

**a.)**

All this does is convert the dots and dashes into something the computer can interpret, the 2 high bits tell the number of dots or dashes where the other bits contain the 1 and zeros

**b.)**

I connected the RGB leds to pin 8,9 and 10 so that we could conserve what is hardcoded in the .h file where red is (1<<0), I configured them as outputs by

#define DDRB\_Reg (\*((volatile uint8\_t \*) 0x24)) // I/O register, if set high the state is an output

DDRB\_Reg |= LED\_RED;

How I turned them on and off

PORTB\_Reg |= led;

PORTB\_Reg &= ~led;

Where led is the parameter passed into the function

**c.) the rest on vs code homework 3**