Assuming that each LED has a 2V drop across it when it is lit, what is the maximum amount of current that will be sourced or sunk through each connected I/O pin on the Arduino/ATmega328 when an LED segment is turned on? (Please look at the resistance value of an in-series current limiting resistor with an LED on the back of the board).

…. Assuming that the anode is supplied a fixed 5 V (from the USB cable connecting to the PC), and each LED has a 2 V drop across it when it is lit up (not counting the "point" segment, so only 7 segment LED's are used), the maximum total current that would flow into the Arduino/ATmega328 when one LED segment is turned on is 3 mA. This can be proved using Nodal Analysis at each Non-essential Node following each LED:

$$I_{LED_a} = \frac{5\,V - 2\,V}{1\,k\Omega} = 3\,mA$$

Referring to Section 32.1 "Absolute Maximum Ratings" of the ATmega328 datasheet found on the Atmel website, the maximum DC Current per I/O Pin is 40 mA (ATmega328/P, 2016, p. 365). So, 3 mA per IO pin is safe.

So if all of the 7 segments are on, the maximum amount of current that will be sourced or sunk through each connected IO pin on the Arduino/ATmega328 is 21 mA.

$$I_{TOTAL} = \left[\frac{5\,V - 2\,V}{1\,k\Omega}\right] \times 7 = 21\,mA$$

If every (16 total) segment is turned on at the same time, and the sum of all of these currents had to be sunk through the microcontroller simultaneously, what percentage of the total current able to be sourced/sunk by the ATmega328 would be consumed by just these LEDs (refer to the Absolute Maximum Ratings section of the ATmega328 datasheet)?

If all of the 16 segments (including both decimal points) were turned on at the same time, then the sum of all the currents sunk through the microcontroller simultaneously is 48 mA.

$$I_{TOTAL} = \left[\frac{5\,V - 2\,V}{1\,k\Omega}\right] \times 16 = 48\,mA$$

Again, referring to Section 32.1 "Absolute Maximum Ratings" of the ATmega328 datasheet found on the Atmel website, the maximum DC Current for the VCC (power input) and GND pins is 200 mA (ATmega328/P, 2016, p. 365).

Therefore, 24% of the total current able to be sourced of sunk by the ATmega328 would instead be consumed by just the LED's.

$$\%\ of\ Current\ Consumed\ by\ LED's = \frac{48\,mA}{200\,mA} \times 100 = 24\%$$

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

Method 1: Direct LED Segment Access

Question 1 for Lab Report) What does the right-hand digit of the dual 7-segment display show? Record this for your report. A flashing 3 is displayed.

Question 2 for Lab Report) Use the LED display's datasheet to figure out what you *expect* the voltage to be at pins a - g on the right digit, then measure each pin's voltage and document your findings. Using the LED display's datasheet, the expected voltages at pins a through g are listed in the following table (LTD-2601G series, 2000, p. 3). If an LED is OFF, the DMM would show + 5V, but if an LED is ON, the DMM would show close to 0V (because of CA, active low configuration, meaning to turn a LED on, give it 0 V or give it a bit value of 0). So, for the flashing 3 to be displayed:

Pin	Expected Voltage [V]	Measured Voltage [V]
a	0 V	0 V
b	0 V	0 V
c	0 V	0 V
d	0 V	0 V
e	5 Volts	5.13 V
f	5 Volts	5.13 V
g	0 V	0 V

Modify the sketch to alternately display '2' then '4' on the right-hand digit instead of whatever the sample sketch displays. This should give you a sense for what is involved in displaying patterns directly on a 7-segment LED using the Arduino library functions. (Question 3 for Lab Report) Call your sketch *direct24_ardustyle* and include it in your lab report. Modified direct_ardustyle. Modified portions are highlighted in yellow. Make sure to put a delay between the 2 and the 4; otherwise you won't be able to see the 2 because it's moving too fast.

```
/*
 * Digital I/O (dual 7-segment display board) lab sketch.
 *
 * This sketch demonstrates the use of a 7-segment display using direct-access to
 * each of the digit's segments.  The function of this program is left as an exercise
 * to the lab student.
 *
 * The arrangement of the digit's LED segments "a" through "g" are documented in the
 * datasheet for the display.  This display is a common-anode type unit.
 *
 * v1.00 Eric B. Wertz 2011/09/19 00:03 - Initial revision
 */

// Arduino digital pin numbers for right-digit direct segment access
#define RDIGIT_DIRECT_A  8
#define RDIGIT_DIRECT_B  7
#define RDIGIT_DIRECT_C  6
#define RDIGIT_DIRECT_D  5
#define RDIGIT_DIRECT_E  4
#define RDIGIT_DIRECT_F  3
#define RDIGIT_DIRECT_G  2

// min/max pin numbers for looping through all the digit's pins (assumes contiguous
pin numbers!)
#define RDIGIT_DIRECT_MIN 2
#define RDIGIT_DIRECT_MAX 8

#define DELAY_BLINK_MSECS 500 // milliseconds between blinks

void rightDigitOff()
{
  byte pin;

  for (pin=RDIGIT_DIRECT_MIN; pin<=RDIGIT_DIRECT_MAX; pin++) {
    digitalWrite(pin, HIGH);
  }
```

```
}

void setup()
{
  byte pin;

  for (pin=RDIGIT_DIRECT_MIN; pin<=RDIGIT_DIRECT_MAX; pin++) {
    pinMode(pin, OUTPUT);
  }
}

void loop()
{
  rightDigitOff();

  delay(DELAY_BLINK_MSECS);

  digitalWrite(RDIGIT_DIRECT_A, LOW);
  digitalWrite(RDIGIT_DIRECT_B, LOW);
  digitalWrite(RDIGIT_DIRECT_C, HIGH);
  digitalWrite(RDIGIT_DIRECT_D, LOW);
  digitalWrite(RDIGIT_DIRECT_E, LOW);
  digitalWrite(RDIGIT_DIRECT_F, HIGH);
  digitalWrite(RDIGIT_DIRECT_G, LOW);

  delay(DELAY_BLINK_MSECS);
  rightDigitOff();
  delay(DELAY_BLINK_MSECS);

  digitalWrite(RDIGIT_DIRECT_A, HIGH);
  digitalWrite(RDIGIT_DIRECT_B, LOW);
  digitalWrite(RDIGIT_DIRECT_C, LOW);
  digitalWrite(RDIGIT_DIRECT_D, HIGH);
  digitalWrite(RDIGIT_DIRECT_E, HIGH);
  digitalWrite(RDIGIT_DIRECT_F, LOW);
  digitalWrite(RDIGIT_DIRECT_G, LOW);

  delay(DELAY_BLINK_MSECS);
}
```

Overhaul the sketch to use register-style programming to display the number 5 on the right-hand digit. You may not use any Arduino functions, only the DDRx, PORTx and PINx registers and the C-language bit-wise operators. Call your sketch *direct5_regstyle* and (Question 4 for Lab Report) include it in your lab report. You may use the code template *direct5_regstyle.ino* on the course website to get started. Used direct5_regstyle as template.

Digital Pin 1 == TX == Transmitting == setup as OUTPUT (1).
Digital Pin 0 == RX == setup as INPUT (0).

```
/*
 * Digital I/O (dual 7-segment display board) lab sketch.
 *
 * This sketch displays the number "5" on the right-hand digit on the
 * display board using only Register-style programming.
 *
 * Segment "a" is connected to PORTB:0, and "b" through "g" are connected to
 * PORTD:7 through PORTD:2, respectively.  We are avoiding using PORTD:1-0
 * because the serial port is connected to them, and using them can prevent
 * the Serial Console or code download from working.
```

```
 *
 * Remember that the display is a common-anode type unit, so each segment is
active-low.
 *
 *      PORTB    PORTD      <- PORT
 *    --543210  76543210    <- bit number
 *    -------a  bcdefg--    <- segment letter
 *
 * v1.00 <your name here> <current time/date here>
 */

/* millis() function returns the number of milliseconds (an
   unsigned long) since the board started running its current sketch
   previousMilliseconds will store the prev time LED was toggled (t1)
   currentMilliseonds will store the current time (t2)
 */
unsigned long previousMilliseconds = 0;
unsigned long currentMilliseconds = 0;

void setup()
{
  DDRB = (B00000001);
  DDRD = (B11111110);
}

void loop()
{
  /* Check to see if it's time to blink the LED: if the difference
     between the current time and last time you blinked the LED
     is bigger than the interval at which you want to blink the LED.
   */
  currentMilliseconds = millis();

  /*  1) IF STATEMENT TO DELAY, THEN DISPLAY A 5  */
  /* If you have reached 100 ms
     If t2 - t1 is >= 100 ms
   */
  if(currentMilliseconds - previousMilliseconds >= 100)
  {
    /* Store the current time into the previousMilliseonds variable
       to subtract later in each subsequent pass of the if-statement
     */
    previousMilliseconds = currentMilliseconds;

    PORTB = (B11111110);
    PORTD = (B10010011);
  }

  currentMilliseconds = millis();

  /*  2) IF STATEMENT TO DELAY, THEN TURN OFF LED's  */
  if(currentMilliseconds - previousMilliseconds >= 100)
  {
    /* Store the current time into the previousMilliseonds variable
       to subtract later in each subsequent pass of the if-statement
     */
    previousMilliseconds = currentMilliseconds;

    PORTB = (B11111111);
    PORTD = (B11111111);
  }
}
```

Test this function to complete a sketch to count down continuously from 9 to 0 with one-second pauses in between. (**Question 5 for Lab Report)** **Include this test program which also contains the `rightDigitDirect()` function in your lab report.**

```
/*  I) Function Prototypes  */
void rightDigitDirect(int digit);

/*  II) setup()  */
void setup()
{
    DDRB = (0b00000001);
    DDRD = (0b11111110);
}

/*  III) loop() is analogous to main() */
void loop()
{
    for(int i = 0; i < 10; i++)
    {
      rightDigitDirect(i);
    }

    delay(1000);
    PORTB = (B11111111);
    PORTD = (B11111111);

    for(int j = 9; j >= 0; j--)
    {
      rightDigitDirect(j);
    }

    delay(1000);
    PORTB = (B11111111);
    PORTD = (B11111111);
}

/*  IV) Function Definitions  */
void rightDigitDirect(int digit)
{
    switch (digit)
    {
      case 0:
        //display 0 when digit equals 0
        // 0
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B00000111);
        break;
      case 1:
        //display 1 when digit equals 1
        // 1
        delay(1000);
        PORTB = (B11111111);
        PORTD = (B00111111);
        break;
      case 2:
        //display 2 when digit equals 2
        // 2
```

```
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B01001011);
        break;
      case 3:
        //display 3 when digit equals 3
        // 3
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B00011011);
        break;
      case 4:
        //display 4 when digit equals 4
        // 4
        delay(1000);
        PORTB = (B11111111);
        PORTD = (B00110011);
        break;
      case 5:
        //display 5 when digit equals 5
        // 5
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B10010011);
        break;
      case 6:
        //display 6 when digit equals 6
        // 6
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B10000011);
        break;
      case 7:
        //display 7 when digit equals 7
        // 7
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B00111111);
        break;
      case 8:
        //display 8 when digit equals 8
        // 8
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B00000011);
        break;
      case 9:
        //display 9 when digit equals 9
        // 9
        delay(1000);
        PORTB = (B11111110);
        PORTD = (B00110011);
        break;

      default:
        // if nothing else matches, do the default
        // default is optional
       break;
  }
}
```

Method 2: 7447 BCD-to-7-segment (CA) display driver

(Question 6 for Lab Report) What advantage(s) and disadvantage(s) are there with using a dedicated decoder chip like the 7447 with a 7-segment LED display, versus driving the LED segments directly from Arduino pins? Consider hardware complexity, software complexity, power requirements, wiring, available I/O, etc.

Several advantages arise from using a dedicated decoder chip such as the 7447 on a 7-segment LED display. There is one less port to initialize: now, only PORTC will be used instead of PORTB and PORTD as with the direct LED segment access. Header pins ABCD are used instead of a through g. So, there are 3 less wires. The bit patterns are thus shorter: 4 bits instead of 8. Therefore, the 7447 can drive the 7-segment to display up to the number 15 using only one line it pattern. However, a disadvantage is that the 7447 uses Analog pins on the Arduino UNO, rather than with Digital pins, which seem to be easier to handle.

Advantages of Arduino Style IO: The Arduino style of pin control is very effective and easy to understand at a simple level, by using port style programming, the simplicity of these functions is replaced by speed and versatility.

Disadvantages of Arduino Style IO: That being said, using Arduino style control is very user friendly and readable and easy to implement in code. While the Arduino style functions like digitalWrite() are nice, they are slow when compared the reaction speed of port style IO, and are limiting the capabilities of the microcontroller for more advanced users.

The advantage of the using a dedicated decoder chip like the 7447 with a 7-segment display is the reduction of required pins to control the display. Contrasting to the primitive way of directly connecting the cathode of the LEDs to the Arduino, the decoder only requires 4 pins to control the 7 segments LED. Having an interface also could increase the overall power draw for the display as the 200mA limit for the Arduino IO pins would not apply anymore. However, the trouble would be increasing the complexity of the control of the display and the need of sourcing the right chip.

Using the 5V power and ground source from the Arduino, use wire jumpers on the breadboard to tie all four of the ABCD pins for the decoder (as shown in Figure 9) on the display board to ground. (Question 7A for Lab Report) What number is displayed on the left-hand digit of the display? When pins ABCD are tied to GND, a 0 is displayed on the left-digit. This corresponds to Appendix A's truth table, which states that for the left-digit (active high), pins D C B A have to be low (i.e. 0 V, or a bit value of 0).

Change the A pin from GND to 5V. (Question 7B for Lab Report) Now what number is displayed on the display? Next, the A pin was moved from GND to 5 V. When pins BCD are tied to GND and A is tied to 5 V, a 1 is displayed on the left-digit.

Leaving A on 5V, also change the B pin from GND to 5V. (Question 7C for Lab Report) What is this third number being displayed? *Note that this technique can/should be used to validate hardware before trying to use it from the Arduino as this helps eliminate the possibility of "hardware issues" if "nothing seems to be working". This same test procedure could have been done to check the "a" through "g" segments in the direct-access exercises, by the way.* The B pin was then also moved from GND to 5 V. When pins CD are tied to GND and A and B are tied to 5 V, a 3 is displayed.

Disconnect the four hard-wired jumpers to the ABCD pins, and use wire jumpers to connect Arduino *Analog* Pin 0 to display board header pin A (still as in Figure 9). Connect *Analog* Pin 1 to header pin B, *Analog* Pin 2 to header pin C, and *Analog* Pin 3 to header pin D. Download *decoder_ardustyle.ino* from the course website and run it on your Arduino (Question 8A for Lab Report) What does the left-hand digit of the display show? (If you see no number, you have a problem that you need to remedy before you continue.) A 2 is displayed and remains displayed.

(Question 8B for Lab Report) Use the datasheet to figure out what you expect the voltage at pins 9, 10, 11, 12, 13, 14 and 15 *of the 7447 chip* (NOT the *Arduino* pins 9-15) to be, and document your findings. Using the 7447 BCD-to-Seven-Segment Decoder datasheet, the expected voltages at pins 9, 10, 11, 12, 13, 14, and 15 of the 7447 chip are listed in the following table (SN54LS49, 1988, p. 9). The voltages at pins 9, 10, 11, 12, 13, 14, and 15 were then measured and are also documented in the following table.

Pin	Expected Voltage [V]	Measured Voltage [V]
9	0.8 – 2.0	0.159
10	0.8 – 2.0	0.1607
11	0.8 – 2.0	7.4
12	0.8 – 2.0	0.165
13	0.8 – 2.0	0.165
14	0.8 – 2.0	0.1667
15	0.8 – 2.0	3.915

Modify the program to display '6' on the left-hand digit. (Question 9A for Lab Report) Which pins of the 7447 do you expect to be at 5V, and which are low when '6' is displayed? Modified decoder_ardustyle.ino. Modified portions highlighted in yellow. Look at Appendix A truth table to see how to display a 6 on the left-hand digit, which is very straightforward (left hand digit is active high, so LOW means turn off, or give it 0 V, or assign it a bit value of 0 to turn bit OFF); simply have to write D =LOW; C = HIGH; B = HIGH; and A = LOW, just as it says in Appendix A.

Pin 1 (B) and Pin 2 (C) will be at +5 V because they are written HIGH (active high means either giving it +5 V, writing something HIGH, or giving it a bit value of 1 will turn it ON)

Pin 6 (D) and Pin 7 (A) will be at 0 V

```
/*
 * Digital I/O (dual 7-segment display board) lab sketch.
 *
 * This sketch demonstrates the use of a 7-segment display using a BCD-to-7segment
 * decoder to drive a digit's segments.
 *
 * The connections between the Arduino pins and ABCD header pins on the display
 * board are as follows:
 *   Analog Pin 0 (Digital Pin 14) -- display header pin A
 *   Analog Pin 1 (Digital Pin 15) -- display header pin B
 *   Analog Pin 2 (Digital Pin 16) -- display header pin C
 *   Analog Pin 3 (Digital Pin 17) -- display header pin D
 *
 * The function of this program is to be determined by the lab student.
 *
 * v1.00 Eric B. Wertz 2011/09/19 02:10 - Initial revision
```

```
 */

#define LDIGIT_DECODER_A 14
#define LDIGIT_DECODER_B 15
#define LDIGIT_DECODER_C 16
#define LDIGIT_DECODER_D 17

void setup()
{
  pinMode(LDIGIT_DECODER_A, OUTPUT);
  pinMode(LDIGIT_DECODER_B, OUTPUT);
  pinMode(LDIGIT_DECODER_C, OUTPUT);
  pinMode(LDIGIT_DECODER_D, OUTPUT);
}

void loop()
{
  /*  Display a 6  */
  digitalWrite(LDIGIT_DECODER_D, LOW);
  digitalWrite(LDIGIT_DECODER_C, HIGH);
  digitalWrite(LDIGIT_DECODER_B, HIGH);
  digitalWrite(LDIGIT_DECODER_A, LOW);

  delay(500);

  /*  Display a 9  */
  digitalWrite(LDIGIT_DECODER_D, HIGH);
  digitalWrite(LDIGIT_DECODER_C, LOW);
  digitalWrite(LDIGIT_DECODER_B, LOW);
  digitalWrite(LDIGIT_DECODER_A, HIGH);

  delay(500);
}
```

(Question 9B for Lab Report) Experiment by displaying numbers between 0 and 9 on the different digits until you are satisfied with your understanding of what is happening between the Arduino pins, the decoder chip inputs/outputs, and the LED display. Modified to display a 6 then a 9.

Download *decoder_regstyle_static.ino* from the course website and use it as a template to create a sketch that also displays the number 6 on the left-hand digit, *but using only Register-style* programming. (Question 10 for Lab Report) Include your sketch in your lab report.

```
void setup()
{
    DDRC |= 0b1111;           // decoder pins: bottom four bits of PORTC are outputs
    //DDRC |= 0b00001111;     // Alternatively
}

void loop()
{
    PORTC |= 0b0110;
    // PORTC |= 0b00000110;  // Alternatively
}
```

Write a sketch that implements a *function* called leftDigitDecoder() that takes an *int* valued 0 through 9 as a parameter, and displays that digit on the left-hand digit of the display, similar to how rightDigitDirect() worked. Use this function to complete a sketch to count down continuously from 9 to 0 with one-second pauses in between

```
/*  I) Function Prototypes  */
void leftDigitDecoder(int val);

/*  II) setup() is analogous to Constructors  */
void setup()
{
  DDRC = (B1111);
}

/*  III) loop() is analogous to main() */
void loop()
{
  for (int i = 0; i<10; i++)
  {
    leftDigitDecoder(i);
  }

  delay(1000);
  PORTC = (B1001);

  for (int j = 9; j>=0; 9--)
  {
    leftDigitDecoder(i);
  }

  delay(1000);
  PORTC = (B0000);
}

/*  IV) Function Definitions  */
void leftDigitDecoder(int val)
{
    switch (val)
    {
      case 0:
        //do something when var equals 0
        // 0
        delay(1000);
        PORTC = (B0000);
        break;
      case 1:
        //do something when var equals 1
        // 1
        delay(1000);
        PORTC = (B0001);
        break;
      case 2:
        //do something when var equals 2
        // 2
        delay(1000);
        PORTC = (B0010);
        break;
      case 3:
        //do something when var equals 3
        // 3
        delay(1000);
```

```
        PORTC = (B0011);
        break;
      case 4:
        //do something when var equals 4
        // 4
        delay(1000);
        PORTC = (B0100);
        break;
      case 5:
        //do something when var equals 5
        // 5
        delay(1000);
        PORTC = (B0101);
        break;
      case 6:
        //do something when var equals 6
        // 6
        delay(1000);
        PORTC = (B0110);
        break;
      case 7:
        //do something when var equals 7
        // 7
        delay(1000);
        PORTC = (B0111);
        break;
      case 8:
        //do something when var equals 8
        // 8
        delay(1000);
        PORTC = (B1000);
        break;
      case 9:
        //do something when var equals 9
        // 9
        delay(1000);
        PORTC = (B1001);
        break;

      default:
        // if nothing else matches, do the default
        // default is optional
      break;
  }
}
```

Extra Credit: Using the `leftDigitDecoder()` function that you just wrote along with the `rightDigitDirect()` function from the direct-access exercise, write a sketch that counts continuously up from 00 to 99 with 100msec pauses in between numbers. ***Hint***: You will probably find the '/' and '%' C-language operators useful for this exercise. If you don't understand how they work, search the Arduino reference pages for the *modulo* function.

```
/*  I) Function Prototypes  */
void rightDigitDirect(int digit);
```

```
void leftDigitDecoder(int val);

/*  II) setup()  */
void setup()
{
    DDRB = (0b00000001);
    DDRD = (0b11111110);
    DDRC = (B1111);
}

/*  III) loop() is analogous to main() */
void loop()
{
    for (int i = 0; i<10; i++)
    {
        leftDigitDecoder(i);

        for (int j = 0; j<10; j++)
        {
            rightDigitDirect(j);
        }
    }

    delay(1000);

    for (int k = 9; k>=0; k--)
    {
        leftDigitDecoder(k);

        for (int l = 9; l>=0; l--)
        {
            rightDigitDirect(l);
        }
    }

    delay(1000);
}

/*  IV) Function Definitions  */
void rightDigitDirect(int digit)
{
    switch (digit)
    {
      case 0:
        //display 0 when digit equals 0
        // 0
        delay(100);
        PORTB = (B11111110);
        PORTD = (B00000111);
        break;
      case 1:
        //display 1 when digit equals 1
        // 1
        delay(100);
        PORTB = (B11111111);
        PORTD = (B00111111);
        break;
      case 2:
        //display 2 when digit equals 2
        // 2
        delay(100);
        PORTB = (B11111110);
        PORTD = (B01001011);
```

```
        break;
      case 3:
        //display 3 when digit equals 3
        // 3
        delay(100);
        PORTB = (B11111110);
        PORTD = (B00011011);
        break;
      case 4:
        //display 4 when digit equals 4
        // 4
        delay(100);
        PORTB = (B11111111);
        PORTD = (B00110011);
        break;
      case 5:
        //display 5 when digit equals 5
        // 5
        delay(100);
        PORTB = (B11111110);
        PORTD = (B10010011);
        break;
      case 6:
        //display 6 when digit equals 6
        // 6
        delay(100);
        PORTB = (B11111110);
        PORTD = (B10000011);
        break;
      case 7:
        //display 7 when digit equals 7
        // 7
        delay(100);
        PORTB = (B11111110);
        PORTD = (B00111111);
        break;
      case 8:
        //display 8 when digit equals 8
        // 8
        delay(100);
        PORTB = (B11111110);
        PORTD = (B00000011);
        break;
      case 9:
        //display 9 when digit equals 9
        // 9
        delay(100);
        PORTB = (B11111110);
        PORTD = (B00110011);
        break;

      default:
        // if nothing else matches, do the default
        // default is optional
        break;
  }
}

void leftDigitDecoder(int val)
{
    switch (val)
    {
      case 0:
```

```
      //do something when var equals 0
      // 0
      delay(100);
      PORTC = (B0000);
      break;
    case 1:
      //do something when var equals 1
      // 1
      delay(100);
      PORTC = (B0001);
      break;
    case 2:
      //do something when var equals 2
      // 2
      delay(100);
      PORTC = (B0010);
      break;
    case 3:
      //do something when var equals 3
      // 3
      delay(100);
      PORTC = (B0011);
      break;
    case 4:
      //do something when var equals 4
      // 4
      delay(100);
      PORTC = (B0100);
      break;
    case 5:
      //do something when var equals 5
      // 5
      delay(100);
      PORTC = (B0101);
      break;
    case 6:
      //do something when var equals 6
      // 6
      delay(100);
      PORTC = (B0110);
      break;
    case 7:
      //do something when var equals 7
      // 7
      delay(100);
      PORTC = (B0111);
      break;
    case 8:
      //do something when var equals 8
      // 8
      delay(100);
      PORTC = (B1000);
      break;
    case 9:
      //do something when var equals 9
      // 9
      delay(100);
      PORTC = (B1001);
      break;

    default:
      // if nothing else matches, do the default
      // default is optional
```

```
        break;
    }
}
```

(Question 11 for Lab Report) Describe how and why using the Arduino I/O functions (for output, in this case) is better/easier or worse/painful for Exercises 8-10 compared to Exercise 11. Also include any conclusions that you can make about the different code styles' compiled program sizes.

NOTES

NOTES (SPRING 2017)

- Always connect current-limiting resistor when LED is present

Method 1: Direct LED Segment Access (Right Hand Digit)

- Ardustyle (uses Arduino Library Functions)

```
pinMode(RDIGIT_DIRECT_A, OUTPUT);
pinMode(RDIGIT_DIRECT_B, OUTPUT);
pinMode(RDIGIT_DIRECT_C, OUTPUT);
pinMode(RDIGIT_DIRECT_D, OUTPUT);
pinMode(RDIGIT_DIRECT_E, OUTPUT);
pinMode(RDIGIT_DIRECT_F, OUTPUT);
pinMode(RDIGIT_DIRECT_G, OUTPUT);

digitalWrite(RDIGIT_DIRECT_A, HIGH);
digitalWrite(RDIGIT_DIRECT_B, HIGH);
digitalWrite(RDIGIT_DIRECT_C, HIGH);
digitalWrite(RDIGIT_DIRECT_D, HIGH);
digitalWrite(RDIGIT_DIRECT_E, HIGH);
digitalWrite(RDIGIT_DIRECT_F, HIGH);
digitalWrite(RDIGIT_DIRECT_G, HIGH);
```

- Port-style

```
DDRB = (B1111 1111);
DDRD = (B1111 1111);

PORTB = (B1111 1110);
PORTD = (B0000 0011);
```

Method 2: 7447 BCD-to-7-segment display driver Common Anode (Left Hand Digit)

- Ardustyle

```
pinMode(LDIGIT_DECODER_A, OUTPUT);
pinMode(LDIGIT_DECODER_B, OUTPUT);
pinMode(LDIGIT_DECODER_C, OUTPUT);
pinMode(LDIGIT_DECODER_D, OUTPUT);

digitalWrite(LDIGIT_DECODER_A, LOW);
digitalWrite(LDIGIT_DECODER_B, LOW);
digitalWrite(LDIGIT_DECODER_C, LOW);
digitalWrite(LDIGIT_DECODER_D, LOW);
```

- Port-style

```
DDRC = (B1111);

PORTC = (B0000);
```

NOTES (FALL 2017)

Bitwise Operations Review

- 1 << 1 Effectively multiplying by 2
  1 >> 1 Effectively dividing by 2

- Recall that the following two are equivalent:

```
PORTB |= (1111 1111);
PORTB  = PORTB | (1111 1111);
```

Port-Style Programming

- Direct-Data-Register (DDRx) is analogous to PinMode(pin, OUTPUT); , and they are also placed in setup()
  - In DDR-style, for both the Right Digit and the Left Digit, initializing pins to 1 in setup() sets them up as output, and initializing pins to 0 in setup() sets them up as input.
- Port (PORTx) is analogous to digitalWrite(pin, HIGH); , and they are also placed in loop()
  - ~~Because the LED's in the 7-segment display are active-low, setting the pins 0 through 7 as LOW, i.e. initialized to 1, will keep them off.~~

- The LED's on the Right Digit are connected in common anode (CA) configuration and are thus Active Low, meaning all the anode connections of the LED segments are joined together to logic "1".
  - To turn the bit OFF, initialize the pin HIGH (5 V) in Direct-Style programming or initialize the pin with a value of 1 in Port-Style programming

```
digitalWrite(RDIGIT_DIRECT_A, HIGH);
PORTB = (xxxx xxx1);
```

  - To turn the bit ON, set the pin LOW (0 V) in Direct-Style programming or set the pin with a value of 0 in Port-Style programming.

```
digitalWrite(RDIGIT_DIRECT_A, LOW);
PORTB = (xxxx xxx0);
```

- This is not the case with the Left Digit
  - To turn the bit OFF, initialize the pin LOW in Direct-style programming or initialize the pin with a value of 0

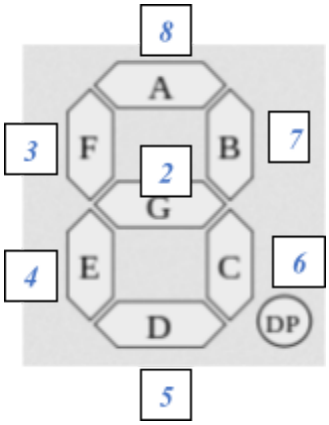
```
PORTC = (0xxx);
digitalWrite(LDIGIT_DECODER_D, LOW);
```

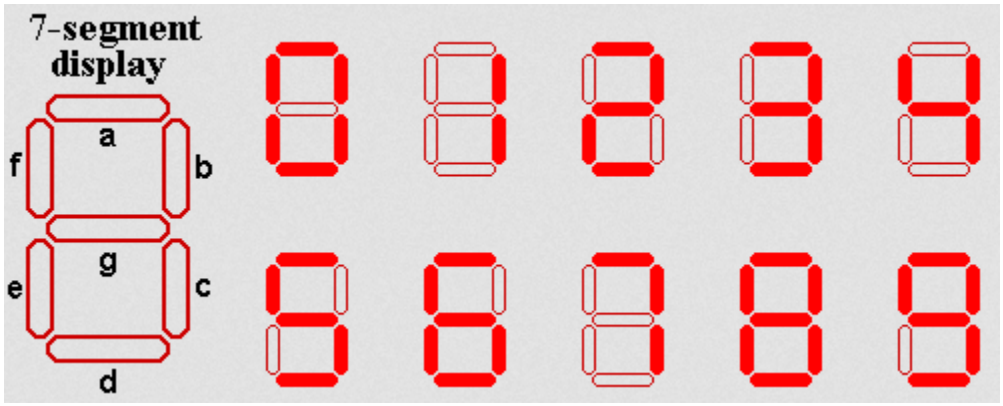
  - To turn the bit ON, set the pin HIGH in Direct-Style programming or set the pin with a value of 1

```
PORTC = (1xxx);
digitalWrite(LDIGIT_DECODER_D, HIGH);
```

- In Port-Style programming, it is best to avoid using Digital Pins 0 and 1 on PORTD because they are used for the Downloading and Programming processes. DP0 and DP1 are on PORTD (the Right Digit). It is best to leave them OFF (leave them with a value of 1) [p. 4].

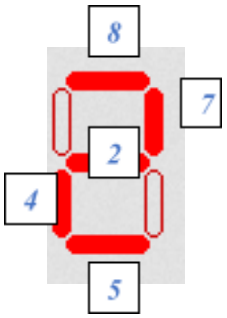
a = Digital Pin 8
b = Digital Pin 7
c = Digital Pin 6
d = Digital Pin 5
e = Digital Pin 4
f = Digital Pin 3
g = Digital Pin 2





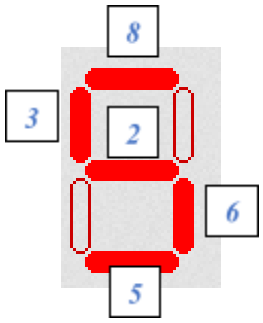
<u>To display a 2 on the Right Digit Display:</u>
Pins 8, 7, 5, 4 and 2 are ON (they have a value of 0)
Pins 6 and 3 are OFF (they have a value of 1).

Also keep pins 15, 14, 13, 12, 11, 10, and 1 and 0 OFF (they have a value of 1) to avoid messing with any other processes that are reserved to those pins (pin 1 and 0 are used for the Downloading and Programming processes).



Pin #								**8**
PORTB=	1	1	1	1	1	1	1	0
PORTD=	0	1	0	0	1	0	1	1
Pin #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

To display a 5 on the Right Digit Display:



Pin #								**8**
PORTB=	1	1	1	1	1	1	1	0
PORTD=	1	0	0	1	0	0	1	1
Pin #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Note: Hexadecimal can also be used (base 16), instead of binary (base 2):

Split the byte (1 byte = 8 bits) into two halves

PORTB=	1	1	1	1
	2^3	2^2	2^1	2^0
	8	4	2	1

8 + 4 + 2 + 1

= 15

0 x F

1	1	1	0
2^3	2^2	2^1	2^0
8	4	2	

8 + 4 + 2

= 14

0 x E

PORTB = 0 x EF;

- Advantage of Port-Style Programming vs. Direct-Style Programming
  - Direct-Style Programming uses Arduino Library Functions, which is what we are used to (from ME 30 and other Arduino programming classes)
  - However, with Port-style programming, it is evident that it takes up much less lines of code. And, you won't risk deleting registers, which may or may not contain an important process or an address.
    - Also, Port-style programming allows for the usage of the bitwise-OR (the single vertical line |). Bitwise-OR allows you to change the value of the desired bits without changing the values of the other bits.
      - For example, `DDRD |= (1111 1100);` forces bits 2 through 7 in PORTD to be a 1 because the bitwise-OR operator works like this:
        - If one of the two bits on either side of the `|` operator is a 1, the result is a 1.
        - However, if both bits are 0, the result is a 0.
    - So, wherever you'd like a 1, put a 1 there and put 0's in the bits you are not worried about.

	Port-Style Programming	Direct-Style (Ardustyle) Programming

Right Digit (1 means output; 1 means turn off LED; 1 means output)	`Setup()`
	`{`
	`  DDRB = (xxxx xxx1);`
	`  DDRD = (1111 1111);`
	`}`
	
	`Loop()`
	`{`
	`  PORTB = (xxxx xxx0);`
	`  PORTD = (0000 0011);`
	`}`
	`// OR:`
	`/*`
	`Setup()`
	`{`
	`  DDRB|= (1111 1111);`
	`  DDRD|= (1111 1111);`
	`}`
	
	`Loop()`
	`{`
	`  PORTB&= (0000 0000);`
	`  PORTD&= (0000 0011);`
	`}`
	`*/`
		`Setup()`
		`{`
		`  pinMode(RDIGIT_DIRECT_A, OUTPUT);`
		`  pinMode(RDIGIT_DIRECT_B, OUTPUT);`
		`  pinMode(RDIGIT_DIRECT_C, OUTPUT);`
		`  pinMode(RDIGIT_DIRECT_D, OUTPUT);`
		`  pinMode(RDIGIT_DIRECT_E, OUTPUT);`
		`  pinMode(RDIGIT_DIRECT_F, OUTPUT);`
		`  pinMode(RDIGIT_DIRECT_G, OUTPUT);`
		`}`
		
		`Loop()`
		`{`
		`  digitalWrite(RDIGIT_DIRECT_A, HIGH);`
		`  digitalWrite(RDIGIT_DIRECT_B, HIGH);`
		`  digitalWrite(RDIGIT_DIRECT_C, HIGH);`
		`  digitalWrite(RDIGIT_DIRECT_D, HIGH);`
		`  digitalWrite(RDIGIT_DIRECT_E, HIGH);`
		`  digitalWrite(RDIGIT_DIRECT_F, HIGH);`
		`  digitalWrite(RDIGIT_DIRECT_G, HIGH);`
		`}`

Left Digit (7447 BCD-to-7-segment display driver) (1 means output; 1 means turn on LED)	`Setup()`
	`{`
	`  DDRC = (1111);`
	`}`
	
	`Loop()`
	`{`
	`  PORTC = (1111);`
	`}`
		`Setup()`
		`{`
		`  pinMode(LDIGIT_DECODER_D, OUTPUT);`
		`  pinMode(LDIGIT_DECODER_C, OUTPUT);`
		`  pinMode(LDIGIT_DECODER_B, OUTPUT);`
		`  pinMode(LDIGIT_DECODER_A, OUTPUT);`
		`}`
		
		`Loop()`
		`{`
		`  digitalWrite(LDIGIT_DECODER_D, HIGH);`
		`  digitalWrite(LDIGIT_DECODER_C, HIGH);`
		`  digitalWrite(LDIGIT_DECODER_B, HIGH);`
		`  digitalWrite(LDIGIT_DECODER_A, HIGH);`
		`}`

How to display a number on the Left Digit as well as the Right Digit (using Appendix A for the Truth Table)

```
PORTB = (0000 0000);
PORTD = (0000 0011);
PORTC = (1111);
```

Appendix A (Truth Table)

Left Digit
(connected to Arduino Analog Pins)

Right Digit
(connected to Arduino Digital Pins)

Analog Pin # — *A3 A2 A1 A0*

8 7 6 5 4 3 2 — *Digital Pin #*

Decimal or Function	Inputs LT	RBI	D	C	B	A	BI/RBO (Note 1)	Outputs a	b	c	d	e	f	g	Note
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H	
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H	
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L	
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L	
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L	
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L	
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L	
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H	(Note 2)
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L	
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L	
10	H	X	H	L	H	L	H	H	H	H	L	L	H	L	
11	H	X	H	L	H	H	H	H	H	L	L	H	H	L	
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L	
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L	
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L	
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H	(Note 3)
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H	(Note 4)
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L	(Note 5)

H = HIGH level, L = LOW level, X = Don't Care

Figure 13. Logical schematic and I/O interpretation of the 7447 chip.

For Inputs (Left Digit)

HIGH == 1 (LED on)

LOW == 0 (LED off)

LOW == 0 (LED on)

HIGH == 1 (LED off)

For Outputs (Right Digit)

Examples:

A3 A2 A1 A0

```
PORTC = (   0   0   0   0  );
```

8

```
PORTB = ( 1  1  1  1  1  1  1  1 );
PORTD = ( 1  1  1  1  1  1  1  1 );
```

7 6 5 4 3 2

Anatomy of an Arduino Program (similar to a C program)

1. Function Prototype (purpose is to have a list of the functions you have in the program; the functions will actually be defined, i.e. filled out in the Function Definitions below)

```
void rightDigitDirect (int val);
```

2. Variables

```
#define LED0 11
#define POTENTIOMETER A2
double average = 0.0;
```

3. Setup ( )

```
void setup()
{

}
```

4. Loop ( )

```
void loop()
{
   for(int i = 0; i < 10; i++)
   {
         rightDigitDirect(i);
   }
}
```

5. Function Definitions (this is where the functions will actually be defined, i.e. filled out)

```
void rightDigitDirect(int val)
{
   switch(val)
   {
         case 0:
            PORTB = (B11111110);
            PORTD = (B00000011);
   }
}
```