Ch-1 Introduction to ATmega328P Microcontroller

1.1 What is an ATmega328P Microcontroller?
It is this 28-pin semiconductor chip of Fig-1.1. The full name is “Micro Controlling Unit”, the short name is “Microcontroller”, and the abbreviated name is “MCU”. The MCU can process only 8-bit data at a time.

![Figure-1.1: Pictorial view, pin diagram, and pin signals of ATmega328P microcontroller](image)

1.2 Use of a Microcontroller
We can use a MCU to build the following instruments:
1. Taxi Meter,
2. Prepaid Electrical Energy Meter (PEM), and
3. Digital Weighing Machine (DWM), etc.

![Figure-1.2: Microcontroller based measuring instruments](image)

1.3 Four Basic Tasks of a Microcontroller
1. Takes command (+, -, *, and /) from the user via keyboard,
2. Takes data from the user via keyboard,
3. Modifies data in a way the user wants, and
4. Delivers the result to the user via 7-segment display device or LCD display device.

1.4 Pin Diagram and Pin Signals of ATmega328P MCU
ATmega328P MCU has 28 pins (Fig-1.1) of which 20 are IO line (also known as port-line or port-pin). An IO (input and output) line is used to connect input device like switch and output device like LED.

A pin is associated with one or more signals. For example: Pin-5 is associated with four signals, and these are — PCINT19, OC2B, INT1, and PD3. The signal PD3 which is outside the parentheses is the default function of the pin. The other three signals (PCINT19, OC2B, and INT1) are the alternate signals, and they can be activated if needed through a process known as software initialization or software configuration. A pin has the following features:
(1) It has a serial number. For example: Pin-5
(2) It is associated with one or more signal. For example: Pin-5 is associated with signal PD3
(3) The signal has a meaning. For example: PD3 stands for ‘Bit-3 of PORTD as Output Line’.
(4) The pin/signal has a direction. For example: PD3 signal is an output line.
(5) The pin/signal has electrical characteristics. For example: PD3 can deliver 20 mA current at 4.20 V.

The words pin and signal will interchangeably refer to the same thing. Thus, saying that “Pin-6” works as IO line is equivalent to saying that “signal PD4” works as IO line. The pins are used to connect input devices (switches, temperature sensor, and etc.) and output devices (7-seg display unit, stepper motor, and etc.) with the MCU. A pin is also known as port-pin or port-line as it exchanges 1-bit data with IO (input and output) devices. In Ch-2, we will study the alternate functions of the IO lines.

1.5 Port Structured Diagram for the Pins of ATmega328P Microcontroller

In Fig-1.1, we observe that the MCU has 20 IO lines spread over various pins with symbolic names: PB0 – PB5; PC0 – PC5; PD0 – PD7. In this Subsection, let us organize the pins so that pins with similar function stay together (Fig-1.3).

Figure-1.3: Port structured diagram of ATmega328P MCU

(2) All the IO lines of the MCU have similar functional characteristics. Therefore, any discussion that will be done on a particular IO line (say, Bit-2 of Port-D) will apply to all other IO lines.

(3) Port-D: It will refer to a port when the directions of its IO lines are not yet determined as to be working as input or output lines. Thus, pd0 – pd7 will refer to bit-0 to bit-7 of Port-D Register.

(4) PORTD: It will refer to a port when the directions of the IO line are configured to work as output lines. Thus, PD0–PD7 (or PORTD0–PORTD7) will refer to bit-0 to bit-7 of PORTD Register.

(5) PIND: It will refer to a port when the directions of the IO lines are configured to work as input lines. Thus, PIND0 – PIND7 will refer to bit-0 to bit-7 of PIND Register.

(6) The direction of an IO line is set as output by executing the following command:
```
pinMode(DPin, OUTPUT); //DPin = digital Pin Connector: 0–7 (PD0–PD7), 8–13 (PB0–PS5), 14/00–19/05 (PC0–PC5)
```

(7) Direction of an IO line is set as input without internal pull-up by executing the following command:
```
pinMode(DPin, INPUT); //DPin-XX is input without internal pull-up
```

(8) The direction of an IO line is set as input with internal pull-up resistor by executing this command:
```
pinMode(DPin, INPUT_PULLUP); //every input line has an internal pull-up resistor (Rip = 20k – 50k) Fig-1.3
In Fig-1.3, LED1 is an output device. It is to be placed on the breadboard along with a series current limiting resistor R1. One side of LED1 would be connected with GND pin of UNO (Fig-1.4) with the help of a jumper wire. The other side of LED1 (actually, the other side of R1) will be connected with DPIn-8 of UNO (Fig-1.4).

In Fig-1.3, it is observed that DPIn-8 is internally connected with pin-14 of MCU which is Bit-0 of Port-B. Bit-0 has been assigned a “signal name” called PB0. Here, the LED1 is connected with an output line named DPIn-8 or PB0; however, the name DPIn-8 (simply 8) would be used in the sketch.

To turn ON LED1, we must send Logic High (LH: 3V – 5V) signal at DPIn-8 by executing the following Arduino Code/command/instruction (Section-1.12, Table-1.1).

```java
digitalWrite(8, HIGH);
```

In Fig-1.3, K1 is an input device. One side of K1 is connected with 5V of UNO (Fig-1.5). The other side of K1 is connected with an input line of MCU which is named as DPIn-A3 or DPIn-17 or PINC3 or Bit-3 of Port-C. However, we will use the name DPIn-A3 in the sketch.

When K1 is not pressed, the logic level of DPIn-A3 is GND (0V) by virtue of pull-down resistor R2. DPIn-A3 will assume LH signal when K1 is pressed; as a result, HIGH (1) will appear at Bit-3 of Port-C.

Careful observation reveals that there is an internal pull-up resistor $R_{ip}$ which can be connected with DPIn-A3 by executing the following command. Let us note that both external pull-down and internal pull-up must not be connected with DPIn-A3 at the same time. Also, note that every input line has its own $R_{ip}$ that could be connected or left unconnected.

```java
pinMode(A3, INPUT_PULLUP); //DPIn-A3 is an input line with internal pull-up connected.
pinMode(A3, INPUT);        //DPIn-A3 is an input line without internal pull-up.
```

To check the closing condition of K1 of Fig-1.3, we may execute the following code lines:

```java
pinMode(A3, INPUT);        //DPIn-3 works as input line with external pull-down connected
bool n = digitalRead(A3);  //if K1 is closed, then LH (5V) will be stored in variable n
if(n == HIGH)
{
    Serial.print("K1 is found at closed condition"); //Message will appear on Serial Monitor
}
else
{
    Serial.print("K1 is found at opened condition"); //Message will appear on Serial Monitor
}
```

The operating frequency of the MCU of UNO Board can be set to either 8 MHz from an internal oscillator or to 16 MHz using an external crystal Y1. Currently, the MCU runs at 16 MHz clock frequency.

1.6 Hardware Modules within ATmega328P MCU

In Fig-1.3, it has been shown that there are 3 digital IO ports inside the MCU. In fact, there are many more hardware modules inside the MCU (Fig-1.4). The names and functions of these modules are given below. Details could be covered in Ch-2 (Architecture).

(1) M1: External Clock Oscillator
(2) M2: Clock Prescaler
(3) M3: Internal Clock Oscillator
(4) M4: UART Serial Communication
(5) M5: 8-Bit and 16-Bit Counter
(6) M6: 8-Bit and 16-Bit Timer
(7) M7: External Interrupt Handler Module
(8) M8: Analog to Digital Converter
(9) M9: Port-B
(10) M10: Port-C
(11) M11: Port-D
(12) M12: Watchdog Timer
(13) M13: AVR RISC MCU/CPU
(14) M14: Code/Flash Memory, Fuse Bits, and Lock Bits
(15) M15: EEPROM Memory, Fuse Bits, and Lock Bits
(16) M16: Control Matrix or Sequence Generator
(17) M17: Static RAM Memory
(18) M18: In System Programming Interface (ISP)
(19) M19: SPI Port Serial Communication
(20) M20: Analog Comparator
(21) M21: I2C/TWI Bus Serial Communication
(22) M22: Pulse Width Modulator
1.7 Arduino UNO Learning Board Using ATmega328P Microcontroller

Fig-1.4 shows the pictorial view of the official version of Arduino UNO Learning Board. The UNO Board (simply UNO) offers friendly environment to learn architecture, programming, and interfacing of ATmega328P Microcontroller to build instruments like Temperature-Humidity Meter, Taxi Meter, Digital Weighing Machine, and Prepaid Electrical Energy Meter. The UNO also offers an excellent means for the understanding of the fundamentals of C/C++ Programming Language through the programming and operation of input/output devices and sensors.

![Arduino UNO Learning Board](image1)

Figure-1.5: Arduino UNO Learning Board

1.8 Arduino UNO Learning System

(1) Arduino UNO Learning System (simply UNO System, Fig-1.6) is just a repackaged version of the UNO Board of Fig-1.5. The purpose of the UNO System is to provide the user with a handsome amount of hardware items so that he can comfortably practice the ATmega328P Microcontroller programming and interfacing. The UNO System contains the following items that are mounted on a plastic plate.

![Pictorial view of Arduino UNO Learning System](image2)

Figure-1.6: Pictorial view of Arduino UNO Learning System
## Parts Needed for Lab Work

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Arduino UNO (1)</td>
</tr>
<tr>
<td>2.</td>
<td>Arduino Nano (1)</td>
</tr>
<tr>
<td>3.</td>
<td>2x16 LCD (4)</td>
</tr>
<tr>
<td>4.</td>
<td>LM355 (2)</td>
</tr>
<tr>
<td>5.</td>
<td>CC-type 7-segment Display (4)</td>
</tr>
<tr>
<td>6.</td>
<td>DS18B20 (2)</td>
</tr>
<tr>
<td>7.</td>
<td>Breadboard 16.5 cm X 5.5 cm (1)</td>
</tr>
<tr>
<td>8.</td>
<td>DMe250 (1)</td>
</tr>
<tr>
<td>9.</td>
<td>BMP280 (1)</td>
</tr>
<tr>
<td>10.</td>
<td>2N7000 MOSFET (6)</td>
</tr>
<tr>
<td>11.</td>
<td>DS3231 RTC (4)</td>
</tr>
<tr>
<td>12.</td>
<td>4x4 keypad (1)</td>
</tr>
<tr>
<td>13.</td>
<td>SG-90 Stepper (1)</td>
</tr>
<tr>
<td>14.</td>
<td>Push Button (4)</td>
</tr>
<tr>
<td>15.</td>
<td>HC-05 Bluetooth (1)</td>
</tr>
<tr>
<td>16.</td>
<td>Green LED (5)</td>
</tr>
<tr>
<td>17.</td>
<td>Red LED (5)</td>
</tr>
<tr>
<td>18.</td>
<td>Male-Male Jumper (40)</td>
</tr>
<tr>
<td>19.</td>
<td>Male-Female Jumper (20)</td>
</tr>
<tr>
<td>20.</td>
<td>Resistors: 2.2k (20), 10 (20)</td>
</tr>
<tr>
<td>21.</td>
<td>Plastic Plate and Plastic Box to hold parts and the Assembled Arduino Learning System</td>
</tr>
</tbody>
</table>

*Figure-1.7: Pictorial view of the parts needed for Lab works*
The Arduino UNO Learning System comes with Edge Connectors, Breadboard, ‘Arduino Integrated Development Environment’ (IDE App., Fig-1.8) and Arduino Serial Monitor (SM, Fig-1.9) which offer the following facilities:

(a) IO devices can be connected with MCU pins using edge connectors of the UNO and the breadboard.
(b) An Interface (IDE of Fig-1.8) to create sketch/program/source codes for the solution of a problem using C++ Programming Language and Arduino’s commands/macros.
(c) Program can be compiled to create binary codes in Intel-Hex formatted file using IDE of Fig-1.9.
(d) Binary codes of the Intel-Hex formatted file can be uploaded into the flash of the MCU using IDE.
(e) Free Library Functions (ready-made routines) are available for programming and operation of commonly used industrial sensors/devices like DS18B20 Temperature Sensor, DS3231 RTC etc.

1.9 Integrated Development Environment (IDE) Application Software

![IDE Interface of UNO Learning System](image)

Figure-1.8: IDE Interface of UNO Learning System

1.10 Serial Monitor (SM)

![Serial Monitor of Arduino UNO Learning System](image)

Figure-1.9: Serial Monitor of Arduino UNO Learning System

A Serial Monitor (Fig-1.7) has an InptBox via which a user can send command/data to the UNO Board. It has an OutputBox via which the user can receive data/results from UNO. The Serial Monitor (SM) has many fields of which only one field is being discussed below — the ‘Line ending tab’. The Line ending tab has the following four options:
(1) **No line ending** — When this option is selected, the SM does not send any code to the UNO after sending the ASCII codes for the characters of the InputBox.

(2) **Newline** — When this option is selected, the SM sends 0x0A to the UNO after sending the characters of the InputBox. In C Language, the character representation of ‘Newline’ is ‘\n’.

(3) **Carriage return** — When this option is selected, the SM sends 0x0D to the UNO after sending the characters of the InputBox. In C Language, the character representation of ‘Newline’ is ‘\r’.

(4) **Both NL & CR** — When this option is selected, the SM sends 0x0D followed by 0x0A to the UNO after sending the characters of the InputBox. In C Language, the code is: ‘\r\n’.

1.11 **Built-in LED (L) of Arduino UNO Board**

(1) **The Circuit Diagram**

![Circuit Diagram](image)

**Figure-1.10: Circuit for the built-in LED (L) of Arduino UNO Board**

(2) **Arduino Sketch/Program to Blink L Continuously at 1 sec Interval**

Let us execute the Arduino App (application) from the Start Menu of the PC and observe that the Arduino IDE (Fig-1.8) has appeared on the Desktop. Let us also observe that the IDE contains the following two empty functions: (The MCU executes the setup() function first and then it executes the loop() function and then the user defined functions as and when called upon.)

```c
void setup()
{
}

void loop()
{
}
```

Tasks that will be executed only once or for a ‘number of times’ will be within the setup() function. Tasks that will be executed repetitively will be kept within loop() function.

In the present program we have the following tasks that will be executed only once and hence they will be under setup() function.

(a) To set the direction of the pb5 line as output.

In the present sketch we have the following tasks that will be executed again and again; hence, they will be under loop() function.

(a) To write logic-high at PB5-pin (DPin-3) to ON the LED (L) of Fig-1.10.

(b) To insert 1sec time delay.

(c) To write logic-low at PB5-pin (DPin-3) to OFF the LED (L) of Fig-1.10.

(d) To insert 1sec time delay.

(e) Goto Step-a to repeat the process.
(3) The Final Sketch

```cpp
void setup()
{
  pinMode(3, OUTPUT);  //to set direction of PB5 line as output
}

void loop()
{
  digitalWrite(3, HIGH); //L is ON
delay(1000);   //1000 ms = 1 sec ; Arduino’s Library Function
digitalWrite(3, LOW); //L is OFF
delay(1000);
} //automatically goes to the beginning because of loop() function; no need for goto or jump.
```

1.12 Instructions/Commands Summary for IO Lines

<table>
<thead>
<tr>
<th>Sn</th>
<th>Instruction</th>
<th>Input Argument</th>
<th>Return Arg.</th>
<th>Example</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pinMode</td>
<td>arg1 = DPin, arg2 = INPUT or OUTPUT or INPUT_PULLUP</td>
<td>Nothing</td>
<td>pinMode (2, OUTPUT);</td>
<td>PD2 works as output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = DPin, arg2 = INPUT, arg3 = INPUT_PULLUP</td>
<td></td>
<td>pinMode (2, INPUT);</td>
<td>Bit-2 of Port-D as input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = DPin, arg2 = INPUT_PULLUP</td>
<td></td>
<td>pinMode (2, INPUT_PULLUP);</td>
<td>Without internal pull-up</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = DPin, arg2 = INPUT_PULLUP</td>
<td></td>
<td>pinMode (2, INPUT_PULLUP);</td>
<td>Bit-2as input with internal pull-up enabled</td>
</tr>
<tr>
<td></td>
<td>digitalWrite</td>
<td>arg1 = DPin, arg2 = HIGH or LOW</td>
<td>Nothing</td>
<td>digitalWrite (2, LOW);</td>
<td>PD2 receives LL Value directly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = DPin, arg2 = (indirect) HIGH or LOW</td>
<td></td>
<td>digitalWrite (2, digitalRead(3));</td>
<td>PD2 receives value that is coming from DPin-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = DPin, arg2 = (indirect) HIGH or LOW</td>
<td></td>
<td>digitalWrite (2, bitRead(x, 5));</td>
<td>PD2 receives value that is coming from bit-5 of x</td>
</tr>
<tr>
<td>3</td>
<td>bitWrite</td>
<td>arg1 = PORT name, arg2 = Bit position, arg3 = HIGH or LOW</td>
<td>Nothing</td>
<td>bitWrite (PORTD, 2, LOW);</td>
<td>PD2 of PORTD Register receives value directly.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = PORT name, arg2 = Bit position, arg3 = HIGH or LOW</td>
<td></td>
<td>bitWrite (PORTD, 2, LOW);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = PORT name, arg2 = Bit position, arg3 = HIGH or LOW</td>
<td></td>
<td>bitWrite (PORTD, 2, LOW);</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>digitalRead</td>
<td>arg1 = DPin</td>
<td>Logic value of DPin</td>
<td>bool n = digitalRead(2)</td>
<td>Reads Logic value of DPin-2 and stores in 1-bit variable n.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = PORT name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = PORT name, arg2 = Bit position, arg3 = HIGH or LOW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>bitRead</td>
<td>arg1 = Source variable, arg2 = Bit position</td>
<td>Logic value of bit</td>
<td>bool n = bitRead (PIND, 2);</td>
<td>Logic value of PIND2 is stored in n.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = Source variable, arg2 = Bit position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = Source variable, arg2 = Bit position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>bitSet</td>
<td>arg1 = Variable, arg2 = Bit position</td>
<td>Nothing</td>
<td>bitSet(PORTD, 2)</td>
<td>PD2 of PORTD becomes HIGH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = Variable, arg2 = Bit position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = Variable, arg2 = Bit position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>bitClear</td>
<td>arg1 = Variable, arg2 = Bit position</td>
<td>Nothing</td>
<td>bitClear(PORTD, 2)</td>
<td>PD2 bit of PORTD assumes LL state</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = Variable, arg2 = Bit position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>arg1 = Variable, arg2 = Bit position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>DDRD = 0xnn</td>
<td>Data Dir. Req.</td>
<td>DDRD = 0x11110000</td>
<td>PD7-PD4 works as output; PIND3-PIND0 as input</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>PORTD = 0xnn</td>
<td></td>
<td>PORTD = 0x4</td>
<td>4F is written into PORTD</td>
<td></td>
</tr>
</tbody>
</table>

Figure-1.11: Table showing the software commands for IO Lines of UNO System

1.13 Data Types supported by Standard Arduino (UNO/NANO/MEGA)

<table>
<thead>
<tr>
<th>Sn</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bool y</td>
<td>y will hold either HIGH or LOW and true or false</td>
</tr>
<tr>
<td></td>
<td>unsigned char y</td>
<td>y is 8-bit positive number : 0 to 255 = 0x00 to 0xFF</td>
</tr>
<tr>
<td></td>
<td>Byte y</td>
<td>y is 8-bit positive number : 0 to 255 = 0x00 to 0xFF</td>
</tr>
<tr>
<td></td>
<td>uint8_t y</td>
<td>y is 8-bit signed number in 2’s compliment form (positive ot negative number)</td>
</tr>
<tr>
<td></td>
<td>int8_t y</td>
<td>Range: -128 to +127 = 0x80 to 0x00 to 0x7F</td>
</tr>
<tr>
<td>2</td>
<td>char y</td>
<td>y is 8-bit signed number in 2’s compliment form (positive ot negative number)</td>
</tr>
<tr>
<td></td>
<td>int16_t y</td>
<td>y is 16-bit positive number : 0 to 65535 = 0x0000 to 0xFFFF</td>
</tr>
<tr>
<td></td>
<td>uint16_t y</td>
<td>y is 16-bit signed number in 2’s compliment form (positive ot negative number)</td>
</tr>
</tbody>
</table>
signed int y;
Range: -32768 to +32767 = 0x8000 to 0x0000 to 0x7FFFF

short y;
int16_t y;
y is a 32-bit positive number: 0 to 4294967295 = 0x00000000 to 0xFFFFFFFF

unsigned long int y;
y is a 32-bit positive number: 0 to 4294967295 = 0x00000000 to 0xFFFFFFFF

yUL;
uint32_t y;
y is a 32-bit signed number in 2's compliment for (positive or negative number)
Range: -2147483648 to +2147483647

signed long y;
long y;
yL;
int32_t y;
y is a 32-bit signed number in 2's compliment for (positive or negative number)
Range: -2147483648 to +2147483647

unsigned long long int y;
y is a 64-bit positive number: 0 to 18446744073709551615 = 0x0000000000000000 to 0xFFFFFFFFFFFFFFF

unsigned long y;
yULL;
uint64_t y;
y is a 64-bit signed number in 2's compliment for (positive or negative number)
Range: -9223372036854775808 to +9223372036854775807 = 0x8000000000000000 to 0x0000000000000000 to 0x7FFFFFFF

signed long long int y;
signed long long y;
long long y;
yLL;
int64_t y;
y is a 64-bit signed number in 2's compliment for (positive or negative number)
Range: -9223372036854775808 to +9223372036854775807 = 0x8000000000000000 to 0x0000000000000000 to 0x7FFFFFFF

float y;
(supported by: UNO/NANO/MEGA)
y is binary32 formatted 32-bit pattern for a single precision floating point number with 23 digits to the right of decimal point and having 5/6 digit accuracy.

double y;
(supported by: DUE)
y is binary64 formatted 64-bit pattern for a double precision floating point number with 53 digits to the right of decimal point and having 15/16 digit accuracy.

0b10101010;
Leading 0b (zero b) indicates that the following number is in base 2.
Serial.print(0b10101010, BIN); //shows: 10101010

0x23;
Leading 0x (zero x) indicates that the following number is in base 16.
Serial.print(0x23, HEX); //shows: 23

014
Leading 0 (zero) indicates that the following number is in base 8 (octal).
Serial.println(014, OCT); //shows: 14
Serial.print(014, DEC); //shows: 12

Figure-1.12: Data types supported by Arduino

1.14 Problem and Solutions

1 Refer Fig-1.3 and then write Arduino Codes/Codes/Commands/Instructions to “Turn ON” LED1.

2 Refer Fig-1.3 and then write Codes to “Turn ON” L (built-in LED of UNO).

3 Write codes to check that K1 of Fig-1.3 is at closed condition.

K1 is an input device. One side of K1 is connected with 5V. The other side is connected with DPin-A3 of UNO. There is a pull-down resistor (R2) attached with DPin-3; as a result, the logic level of DPin-A3 is LOW. When K1 will be closed, LH (5V) will appear at DPin-A3. So, the closed condition of K1 could be known by reading the value of DPin-A3. For codes, see Step-1.5(12).

4 Write codes using for() loop to set the directions of the IO lines of Port-D of Fig-1.3 as output.

5 Write codes to show 2 on the 7-segment display device of the following circuit of Fig-1.13.

Figure: 1.13(a)  Figure: 1.13(b)  Figure: 1.13(c)
There are, in fact, 8 segments in a 7-segment display device and these are (read from right to left): p, g, f, e, d, c, b, and a. A segment turns ON when LH is applied at its anode pin and apply LL at it the CC (cathode cathode) pin.

In Fig-1.13(a), DP0 stands for “Display number 0;" CC7SD stands for “cc-type 7-segment display device”; cc0 stands for “cc-pin of DP0.

In Fig-1.13(a), The segments of the display devices are connected with UNO via IO lines of PORTB and PORTD; where, PORTB drives 6 segments (f, e, d, c, b, a) and PORTD drives 2 segments (p, g).

To see 2 on the display devices, the cc-code of 2 (0101 1011 = p, g, f, e, d, c, b, a = 0x5B) is to be sent to the segment lines of the display; where lower 6-bit will be delivered via PORTB (DPins: 13, 12, 11, 10, 9, 8) and the upper 2-bit will be delivered via PORTD (DPins: 7, 6). And then LL is to be asserted on DPin-A0.

The Codes:

(1) Set directions of DPins: 7-6 and 13-8 as output.

for(int i = 8; i < 14; i++)
{
  pinMode(i, OUTPUT);
}

(2) Send the bits for the cc-code of 2 on DPins: 7-6 and 13-8.

byte x = 0x5B;          //cc-code of 2
PORTB = 0x5B;          //0101 1011 ; lower bits 011000 would be accepted by PORTB (DPins: 13-8)

bool n = bitRead(x, 6);       //bit-6 of x (1) enters into variable n; for bitRead() function see 1.12(5)
digitalWrite(6, n);          //bt-6 of x goes on DPin-6;

n = bitRead(x, 7);
bitWrite(PORT, 7, n);        //bit-7 of x goes on DPin-7; for bitWrite() function see 1.12(3)

6 Data Types

(1) To store 1-bit data like HIGH/LOW and true/false, we declare a variable preceded by keyword bool. Here, “bool” is a data type. For example:

bool n = digitalRead(A3);    //variable n will hold LH if K1 is closed.

(2) To store only positive valued 8-bit data (0 to 255 = 0x00 to 0xFF), we declare a variable preceded by keyword byte. Here, “byte” is a data type. For example:

byte x = 0x31;       //variable x holds the binary number 00110001 (Hex is a compact form of binary)

(3) For other data types, see Section-1.3 Table-1.2.

7 Under what condition, the pb6 and pb7 IO lines are available to the user.

8 MCU is the short name for Microcontroller and Microcontrolling Unit is the full name.

9 Assume that LED3 is connected with PD3 line of Fig-1.3in series with a 2.2k current limiting resistor. Write all possible codes/commands/instructions to ignite LED3?

digitalWrite(3, HIGH);     //digitalWrite(Dpin, bitValue);
bitWrite(PORTD, 3, HIGH);  //bitWrite(nameOfOutputPortRegister, bitPosition, bitValue)
bitSet(PORTD, 3);          //arg1 = Output Port; arg2 = bit position

10 In Arduino Platform, the leading 0 (zero) has an especial meaning to refer to octal base. Explain with examples, the validity of the statement.

Serial.println(14, DEC);   //shows: 14 (decimal 14 appears in decimal base)
Serial.println(014, OCT);   //shows: 14 (octal 14 appears in octal base)
Serial.println(014, DEC);   //shows: 12 (octal 14 appears in decimal base)