Lab Manual

for

Microcontroller Lab

6139

Diploma In Computer Engineering

6^{th} Semester
(2015 Revision)

AKNM GOVT. POLYTECHNIC COLLEGE
THIRURANGADI, MALAPPURAM-676317

GENERAL INSTRUCTIONS

1. Submit both fair and rough records on every lab sessions.

2. Fair record should be very neat and easy to read.

3. Write program on left side of the record.

4. Use left side of the record for drawing Pin out diagrams and Digital circuits. All others are to be in the right side.

5. Use HB pencil for drawing diagrams and circuits.

6. Submit records in time.

EXP #	TITLE	AIM
01	LED INTERFACING - 1	To send one byte of data on to PORTB of ATMEGA32 thereby understanding the operation of PORT as output. Write program in C.
02	LED INTERFACING – 2	To send data from 00 to FF to PORTB of ATMEGA32 by giving sufficient delay between out operation. Write program in C.
03	LED INTERFACING 3	To blink LED connecetd to PORTB of ATMEGA32 ON and OFF at 1sec interval with internal RC oscillator at 1Mhz.
04	BIT-WISE OPERATORS	To blink LED connected to PB1 at 1sec interval while keeping LED on PB7 on using AND, OR operations and with generalized approach.
05	DATA CONVERSION – PACKED BCD TO ASCII	Write an AVR C program to convert a given packed BCD to ASCII and display the bytes on PORTB and PORTC.
06	DATA CONVERSION – ASCII TO PACKED BCD	Write an AVR C program to convert give two ASCII digits into packed BCD and display it on PORTB.
07	DATA CONVERSION – BINARY(HEX) TO DECIMAL	Convert Binary (hex) to decimal and display the digits on PORTB with one second delay between each digit.
08	PORT PIN AS INPUT	To read switches connected to PORTD and output the contents to LEDs connected to PORTB. Write program in C.
09	RELAY INTERFACE	To switch relay connected to PORTC ON/OFF at 1 second interval.
10	7 SEGMENT DISPLAY	To display ‘0123’ on multiplexed seven segment cathode display with segments connected to PB0 to PB6 and digits connected to PD4, PD5,PD6 and PB7 respectively.
11	BIT-WISE OPERATORS IN C	Write AVR C program to perform AND, OR, XOR operations on two constant data and outputs the result to Port B with 5 seconds delay between each result.
12	ARITHMETIC OPERATORS IN C	Write AVR C program to perform addition, subtraction and multiplication operations on two constant data and outputs the result to Ports B with sufficient delay between each.
13	LED INTERFACING- 4 ASSEMBLY	To send one byte of data on to PORTB of ATMEGA32 thereby understanding the operation of PORT as output. Write program in assembly language.
14	LED INTERFACING-5 ASSEMBLY	To send data from 00 to FF to PORTB of ATMEGA32 by giving sufficient delay between out operation. Write program in assembly language.
15	PORT PINS AS INPUT-ASSEMBLY	To read switches connected to PORTD and output the contents to LEDs connected to PORTB. Write program in assembly language.

Experiment No: 1 DATE :

LED INTERFACING – 1

Aim:

Write an AVR C program to send one byte of data to PORTB of ATMEGA32 to understand the operation of PORT as output.

Theory

ATMEGA ports are 8 bit wide. Each port has 3 eight bit registers associated. Each bit in these registers configures pins of associated port. Bit 0 of these registers is associated with Pin 0 of the port, Bit1 of these registers is associated with Pin1 and so on.

These three registers are

– DDRx register

– PORTx register

– PINx register

X may be replaced by A,B,C or D based on the PORT you are using.

DDRx register

DDRx (Data Direction Register) configures data direction of the port pins. Which, writing 0 to a bit in DDRx makes corresponding port pin as input, while writing 1 to a bit in DDRx makes the corresponding port pin as output.

EXAMPLE:

- to make all pins of port B as input, DDRA = 0b00000000;
- to make all pins of port A as output pins : DDRB= 0b11111111;
- to make lower nibble of port B as output and higher nibble as input :
  DDRB = 0b00001111; In hexadecimal representation, it can be written as DDRB = 0x0F;

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit as shown in figure 1.1 and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

Result:

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

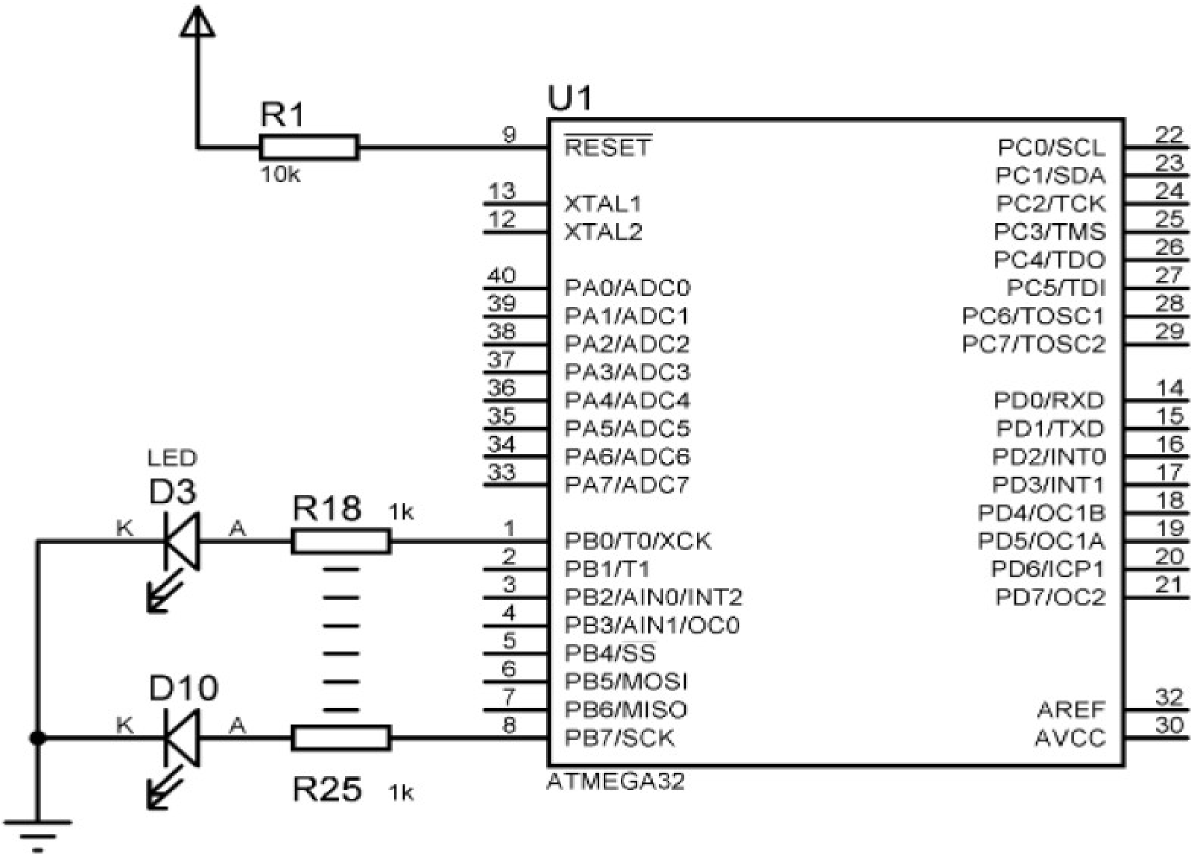


Figure 1.1 : Circuit Diagram

Program

```
/*-----------------------------------------------------------------
This program demonstrates the working of port
Author : WRITE YOUR NAME Date : 15.01.2018

Program set LED connected to PORTB alternately on
PB7 -0 PB6-1 PB5-0 PB4-1 PB3 -0 PB2-1 PB1-0 PB0-1
------------------------------------------------------------------*/

#include <avr/io.h> // standard AVR header

int main(void)
{
	DDRB = 0xff; // setting PORTB as output
	PORTB = 0B01010101; // setting value for PORTB

	while (1); // indefinite loop
}
//-----------------------------------------------------------------
```

Experiment No: 2 DATE :

LED INTERFACING – 2

Aim:

To send data from 00 to FF to PORTB of ATMEGA32 by giving sufficient delay between out operation. Write program in C.

Theory:

A data variable z is defined and initialized with zero, and during each iteration of the for loop, the value of z is given to PORTB and increment z by one. The program never exits the for loop because if you increment an unsigned char variable when it is 0xFF, it will become zero.

Procedure:

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit as shown in Figure 2.1 and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

Result:

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

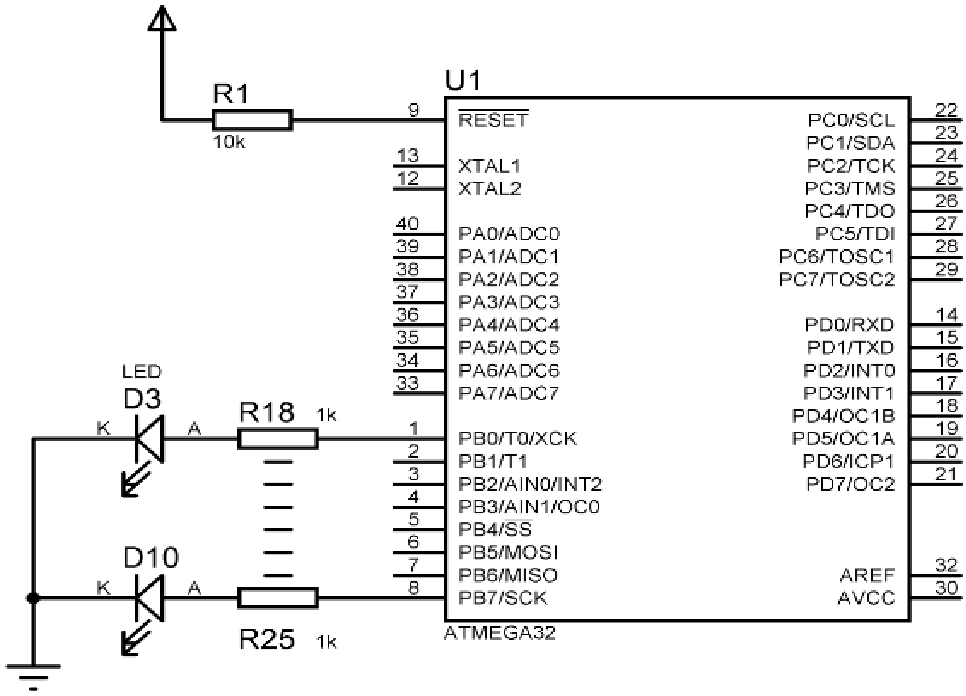


Figure 2.1 Circuit Diagram

Program

```
/* * 02_EXP_LED_C.c * * Created: 17/01/2018 17:40:43 * Author : CTS6 */

/* Write an AVR C pgm to send 00 to FF to Port B */
---------------------------------------------------------------------------------
#include <avr/io.h> // standard AVR header
#include<util/delay.h> // delay header file
#define F_CPU 1000000UL /* it tells the compiler that the MC running at 1MHz. */
int main(void)
{
        unsigned char z; // defining data variable z
        DDRB = 0xFF; // setting PORTB as output

        for(z=0;z<=255;z++) // data z varies from 0 to 255
        {
                        PORTB = z; // sending z to PORTB
                        _delay_ms(10000); // time in ms
        }
        return 0;
}
```

Experiment No: 3 DATE :

LED INTERFACING – 3

Aim:

Write an AVR C program to blink LED connecetd to PORTB of ATMEGA32 ON and OFF at 1sec interval with internal RC oscillator at 1Mhz.

Procedure:

1. Develop the program using AVR Studio, and build its hex file.
2. Burn the hex file into ATMega32 chip using eXtreme Burner software.
3. Observe the result

Theory:

PORTB is set as output and all the bits of PORTB is initialized as ON. A delay of one second is set by using the timer function _delay_ms. After assigning the delay the value in PORTB is negated and assigned to PORTB
by using the command PORTB = ~ PORTB. This two commands are placed in an infinite while loop.

Result:

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working of the embedded system.

Program:

```
/*------------------------------------------------------------------
This program blink LED connecetd to PORTB ON and OFF at 1sec interval
with internal RC oscillator at 1Mhz
author: CTS6Date: 17.01.2018
---------------------------------------------------------------------*/
#define F_CPU 1000000UL
#include <avr/io.h>                     // standard AVR header
#include <util/delay.h>          // delay loop function

int main(void)
{
        DDRB = 0xFF;                    // setting PORTB as output
        PORTB = 0b11111111;             // setting value for PORTB
        while (1)               // indefinite loop
        {
                _delay_ms(1000);
                PORTB = ~ PORTB;
        }
}
//-------------------------------------------------------------------
```

EX.NO. 04 DATE :

BIT-WISE OPERATORS

Aim: To blink LED connected to PB1 at 1sec interval while keeping LED on PB7 on using AND, OR operations and with generalized approach.

Theory:

Previous concept of PORT setting by moving values to the entire port cannot be used in practical cases as moving can change values of pins which are not desired to be affected.

Hence usually AND and OR logic operations are used to manipulate bits.

In order to set PB7 in this case

PORTB |= 0B10000000; // setting PB7

In order to clear PB1

PORTB &= 0B11111101; // clear PB1

This can be rewritten as

In order to set PB7 in this case

PORTB |= (PORTB <<7);

// setting PB7

Or as

PORTB |= (PORTB <<PB7); // setting PB7

PORTB &= ~(PORTB <<1); // clear PB1

Or as

PORTB &= ~(PORTB <<PB1);

// clear PB1

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit as shown in figure 3.1 and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
/*
* 04_EXP_LED_C.c
*
* Created: 07/02/2018 15:03:03
 * Author : Name */
/*
Aim : To blink LED connected to PB1 at 1sec interval while keeping LED on PB7 on
using AND, OR operations and with generalized approach. */
/* Program Blink LED connected to PB1 at 1sec interval while keeping
PB7 always high using bitwise AND, OR operations */

#include <avr/io.h>
#include <util/delay.h>
// standard AVR header
int main(void)
{
	DDRB = 0xff;		// setting PORTB as output
	PORTB |= (1<<PB7); // setting value for PORTB
	while (1)	// indefinite loop
	{
		PORTB |= (1<<PB1);	// setting Pin 1 of PORTB
		_delay_ms(10000);
		PORTB &= ~(1<<PB1);	// resetting Pin 1 of PORTB
		_delay_ms(10000);
	}
}
```

EX.NO. 05 DATE :

PACKED BCD TO ASCII

Aim: Convert Packed BCD 0x29 to ASCII and display the bytes on PORTB.

Theory: To convert packed BCD to ASCII, you must first convert it into unpacked BCD. Then the unpacked BCD is tagged with 011 0000 (30H).

Packed BCD	Unpacked BCD	ASCII
0x29	0x02, 0x09	0x32, 0x39
00101001	00000010, 00001001	00110010, 00111001

Procedure

1. Develop the program using AVR Studio, and build its hex file.

2. Burn the hex file into ATMega32 chip using eXtreme Burner software.

3. Observe the result

Result:

The program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

Copy your Program Here

EX.NO. 06 DATE :

ASCII TO BCD

Aim: Convert ASCII digits “4” and “7” to packed BCD and display them on PORTB.

Theory: To convert ASCII to packed BCD, first convert it into unpacked BCD (to get rid of 3), and then combine the numbers to make packed BCD. 4 and 7 on the keyboard give 34H and 37H respectively. The goal is to produce 47H or “0100 0111”, which is packed BCD.

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Burn the hex file into ATMega32 chip using eXtreme Burner software.
3. Observe the result

Result: The program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

Copy your Program Here

EX.NO. 07 DATE :

BINARY (HEX) TO DECIMAL

Aim: Convert Binary (hex) to decimal and display the digits on PORTB with one second delay between each digit.

Theory: Because the hexadecimal format is a convenient way of representing binary data, we refer to the binary data as hex. The binary data 00-FFH converted to decimal will give 000 to 255. One way to do that is to divide it by 10 and keep the reminder. For example 11111101 or FDH is 253 in decimal. The following is one version of an algorithm for conversion of hex(binary) to decimal.

Hex	**Quotient**	**Reminder**
FD/0A	19	3 (low digit) LSD
19/0A	2	5 (middle digit)
		2 (high digit) MSD

Procedure

1. Develop the program using AVR Studio, and build its hex file.

2. Burn the hex file into ATMega32 chip using eXtreme Burner software.

3. Observe the result

Result: The program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program:

Copy your program Here

Hex to decimal can be generalized as

Copy your second program Here

EX.NO. 08 DATE :

PORT PIN AS INPUT

Aim: To read switches connected to PORTD and output the contents to LEDs connected to PORTB. Write program in C.

Theory:

PINx register

PINx (Port IN) register is used to read data from port pins. In order to read the data from port pin, first you have to change port's data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins. There are two input modes. Either you can use port pins as tri stated inputs or you can activate internal pull up. In order to enable internal pull up, you have to write 1 to the corresponding PORTx register.

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit as shown in figure 8.1 and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

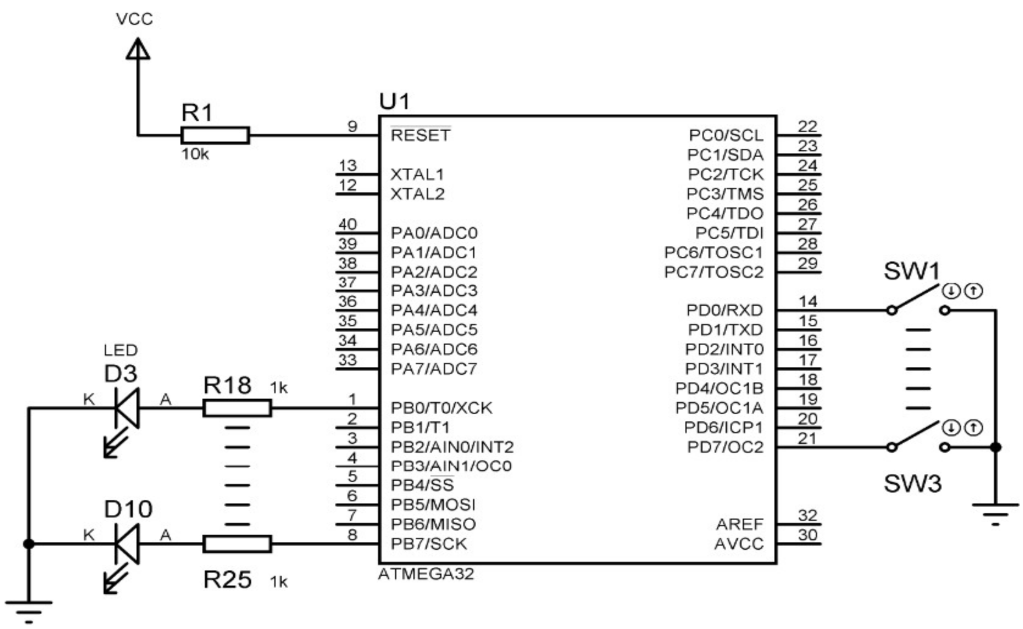


figure 8.1

Program

```
/* * 08_EXP_INPUT_C.c * * Created: 07/02/2018 15:41:01 * Author : USER */
/*---------------------------------------------------------------

This program demonstrates the working of ports as input and output Program reads
switches connected to PORTD and output the values to PORTB
---------------------------------------------------------------*/
#include <avr/io.h>
#include<util/delay.h>
#define F_CPU 1000000UL
int main(void)
{
 DDRB = 0xFF; // DDRB as output
DDRD = 0x00; // DDRD as input
PORTD = 0xFF; //ACTIVATE INTERNAL PULL-UP RESISTORS

while(1)
{
_delay_ms(10); //Calling Delay /* When a key is pressed corresponding bit of PIND
register becomes zero. so to send a HIGH to LED for that key press, we need to give
complement of PIND value to the output port B. */

PORTB = ~PIND;
}
}
```

EX.NO. 09 DATE :

RELAY INTERFACE

Aim: To switch relay connected to PORTC ON/OFF at 1 second interval.

Theory: In most the microcontroller applications, you may need to control external load. This can be done using relays. Relays are electromagnetic switches that are available for various voltages and current ratings. The selection of relays is done based on the current requirement. Usually relays are available at 5, 12, 24 V DC coil voltage. Micro controller port pins can deliver 5V at a very low current rating of 10mA. This power is sufficient only to drive an LED. For driving relays you need to have driver circuit. Drivers are usually transistors in common emitter configuration.

A typical 230V, 6A relay can be driven using the given circuit. For higher current, you have to decrease base resistance. You should remember that any transistor can be driven safely at the rated collector current. For higher current requirement you can choose power transistors as demanded by the case. Diode D acts as free wheel diode.

If you have more devices to control, it is advisable to go for open collector transistor array ULN 2004 or ULN 2003. They can be employed for 12V and 5V applications respectively. In our target board we had provided you with an additional 12V buzzer which is connected to PC5 and relay to PC4 respectively. ULN 2004/3 is provided with inbuilt free wheel diode.

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit as shown in figure 3.1 and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
/*
* 09_EXP_RELAY_C.c * * Created: 07/02/2018 10:55:47 * Author : user */
/*
This program will switch relay and buzzer connected to PORTC 4 and 5
respectively at 1 second interval */
#define F_CPU 8000000UL // clock at 8MHz
#include <avr/io.h> // standard AVR header
#include <util/delay.h> // delay loop function
#define relay PC4
#define buzzer PC5
void InitPort(void)
{
DDRC = 0xFF; // PORTC pins are defined as output
PORTC = 0X00; // Initializing PORTC with 0
}
int main(void)
{
InitPort (); // calling port initialization function
while (1) // indefinite loop
        {
                _delay_ms(3000);
                PORTC |= (1<<buzzer); // buzzer ON
                PORTC |= (1<<relay); // relay ON
                _delay_ms(3000);
                PORTC &= ~(1<<buzzer); // buzzer OFF
                PORTC &= ~(1<<relay); // relay OFF
        }
}
```

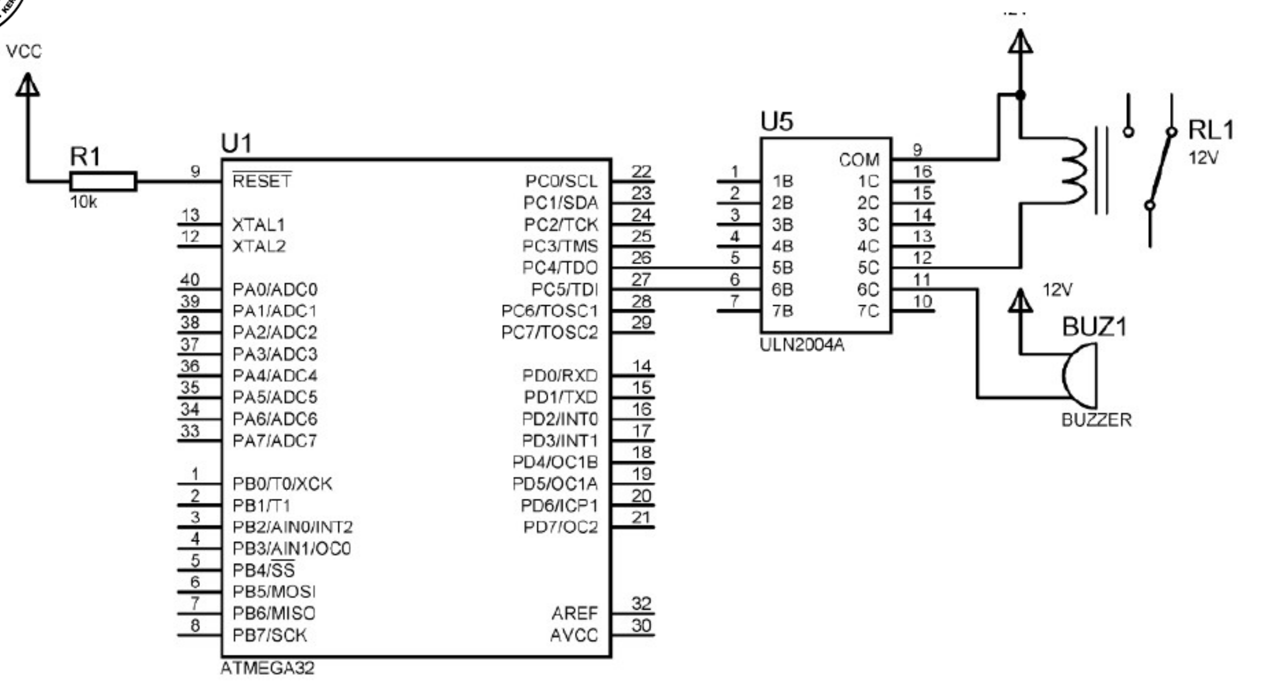


Figure 9.1

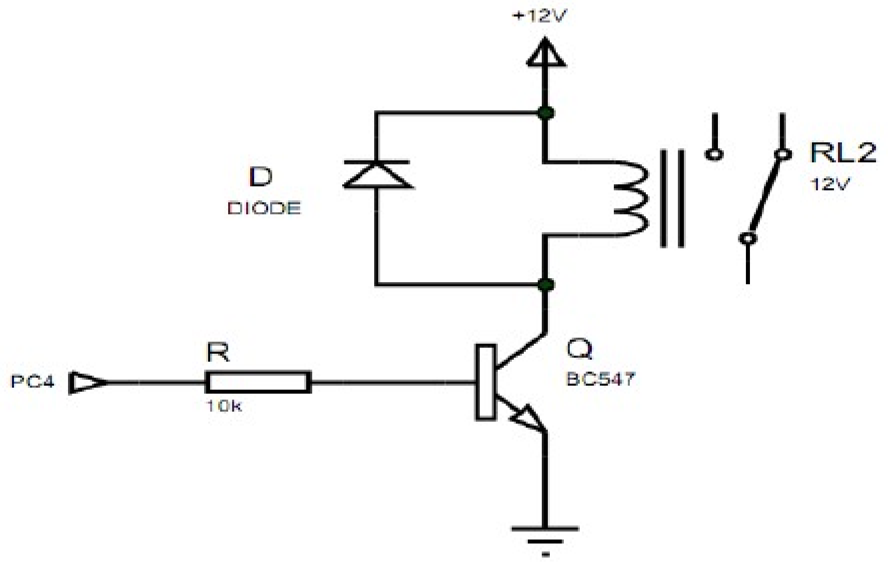


Figure 9.2

EX.NO. 10 DATE :

7SEGMENT DISPLAY

Aim: To display ‘0123’ on multiplexed seven segment cathode display with segments connected to PB0 to PB6 and digits connected to PD4, PD5,PD6 and PB7 respectively.

Theory

Seven segment displays are available in common anode and common cathode configuration. They are usually used in multiplexed configuration. Multiplexing LED displays can make them more efficient by using fewer components, simplifying the printed circuit board, and consuming less power. One segment from each of the digits is connected to a common line which is controlled by a single output pin from the microcontroller. This is done for each segment on a digit, so only seven output drive pins are needed for a multi-digit seven-segment display panel to control all the segments.

One digit is activated at a time using this method, as each digit is enabled one at a time in sequence by a separate signal which connects to the common cathode (or common anode) pin on a digit. There would be a separate control line from the microcontroller output port for each digit on the display panel. The segment data lines are updated for each digit as it is selected by its separate digit select control line. This makes sure that each digit on the panel displays the intended information, which could be a number.

The brightness of a panel is lower on a multiplexed panel because of this on-off duty cycle, but the light intensity can be increased to an acceptable level by a small increase in LED current. LED update frequency can be made high enough so that there wouldn't be any noticeable flickering.

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit as shown in figure 10.1 and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

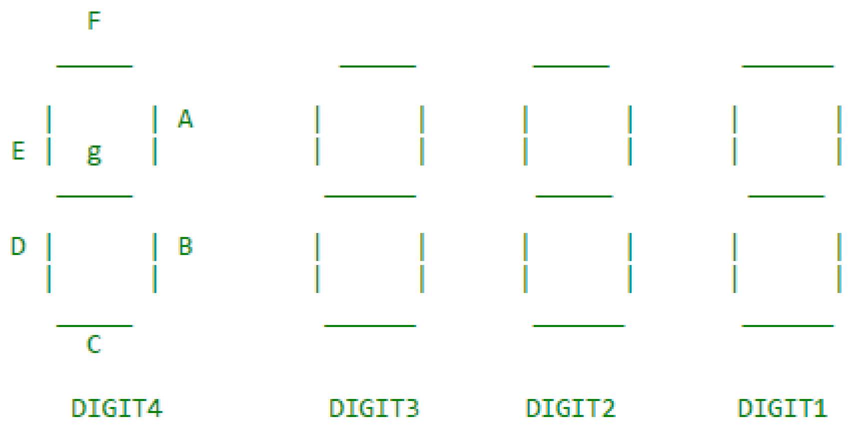
Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
/* * 10_EXP_7SEG_NEW.c * * Created: 07/02/2018 18:56:14 * Author : user*/
 /* Digit select PD4 PD5 PD6 PB7; segment to PB0 TO PB6 */

            /*
```




```
            */

Digit select PD4 PD5 PD6 PB7 segment to PB0 TO PB6

Segments F - PB6 A - PB5 B - PB4 C - PB3 D - PB2 E - PB1 G - PB0

Digit 4 - PD4 Digit 3 - PD5 Digit 2 - PD6 Digit 1 - PB7

----------------------------------------------------------------*/

#define F_CPU 8000000UL // clock @ 8MHz

#include <avr/io.h> // standard AVR header

#include <util/delay.h> // delay loop function



int main(void)

{

    DDRB = 0xff; // setting PORTB as output
```

```
DDRD= 0xff; // setting PORTD as output

PORTD = 0xff; // all set to deselect digit

while (1) // indefinite loop

{

        PORTB = 0x00; // clearing all segments

        PORTD = PORTD | 0xF0; // disabling all digits

        PORTD = PORTD & 0xE0; // selecting digit connected to PD4

        PORTB = 0xFE; // display digit as 0

        _delay_ms(1); // 1 msec delay

        PORTB = 0x00; // clearing all segments

        PORTD = PORTD | 0xF0; // disabling all digits

        PORTD = PORTD & 0xD0; // selecting digit connected to PD5

        PORTB = 0xB0; // display digit as 1

        _delay_ms(1);


        PORTB = 0x00; // clearing all segments

        PORTD = PORTD | 0xF0; // disabling all digits

        PORTD = PORTD & 0xB0; // selecting digit connected to PD6

        PORTB = 0xED; // display digit as 2
```

```
                _delay_ms(1);

                PORTB = 0xFF; // clearing all segments
                PORTD = PORTD | 0xF0; // dissabling all digits
                PORTB = 0x79; // display digit as 3
                _delay_ms(1);

                PORTB = PORTB | 0xFF; // disabling all digit to PB7
        }
}
//--------------------------------------------------------
```

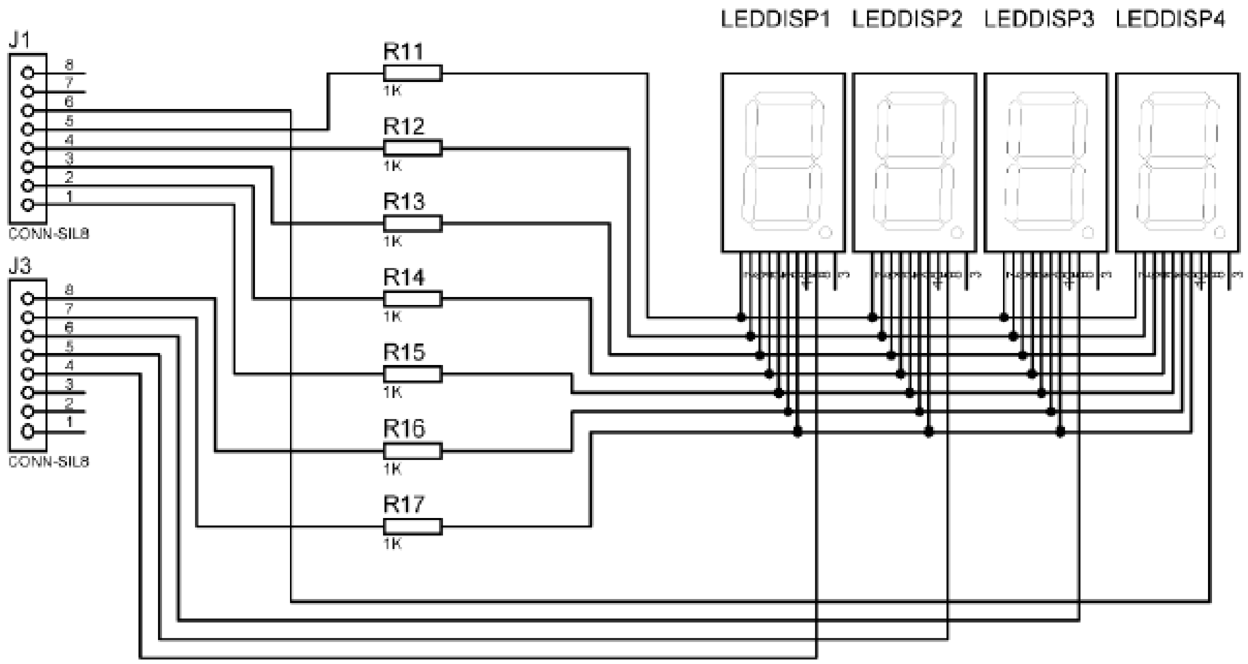


Figure 10.1

EX.NO. 11 DATE :

BIT-WISE OPERATORS IN C

Aim: Write AVR C program to perform AND, OR, XOR operations on two constant data and outputs the result to Ports B with 5 seconds delay between each result.

Theory

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

Table : Some Bit-wise Logic Operators for C

1. 0x35 & 0x0F = 0x05 /* ANDing */
2. 0x04 | 0x68 = 0x6C /* ORing */
3. 0x54 ^ 0x78 = 0x2C /* XORing */
4. ~0x55 = 0xAA /* Inverting 55H */

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
/*
 * Write AVR C program to perform AND, OR, XOR operations
 * on two constant data and outputs the result to Ports B, C and D
 *
 * Created: 18/01/2018 22:11:44
 * Author : ctS6
 */

#include <avr/io.h>
#include<util/delay.h>
#define DATA1 0X35
#define DATA2 0X0F
#define DATA3 0X04
#define DATA4 0X68
#define DATA5 0X54
#define DATA6 0X78

int main(void)
{
      DDRB      =     0xFF;       // make Port B output


      PORTB     =     DATA1 & DATA2;      //ANDing
      _dealy_ms(5000);
      PORTB     =     DATA3 | DATA4;      //Oring
      _dealy_ms(5000);
      PORTB     =     DATA5 ^ DATA6;      //XORing
      _dealy_ms(5000);

   while (1)
   {
   }
      return 0;
}
```

EX.NO. 12 DATE :

ARITHMETIC OPERATORS IN C

Aim: Write AVR C program to perform addition, subtraction and multiplication operations on two constant data and outputs the result to Ports B with 5s delay between each result.

Theory

Some Data Types widely used by C compilers

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648
float	32-bit	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

Procedure

1. Develop the program using AVR Studio, and build its hex file.
2. Implement the circuit and give the power.
3. Burn the hex file into ATMega32 chip using eXtreme Burner software.
4. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
/*
 * EXP# 12
 * ARITHMETIC OPERATORS IN C
 *
 * GccApplication1.c
 *
 * Created: 19/02/2018 06:40:02
 * Author : ctS6
 */

#include <avr/io.h>
#include <util/delay.h>

#define DATA1 7
#define DATA2 3
#define DATA3 0XFF
#define DATA4 0X1
#define DATA5 5
#define DATA6 2

int main(void)
{
    DDRB   =     0xFF;       // make Port B output

    PORTB  =     DATA1 + DATA2;      //ADDITION
    _delay_ms(5000);
    PORTB  =     DATA3 - DATA4;      //SUBTRACTION
     _delay_ms(5000);
    PORTB  =     DATA5 * DATA6;      //MULTIPLICATION
    while (1)
    {
    }
}
```

EX.NO. 13 DATE :

LED INTERFACING - 4

Aim: To send one byte of data on to PORTB of ATMEGA32 thereby understanding the operation of PORT as output. Write program in assembly language.

Theory

i. .include
The .include directive tells the AVR assembler to add the content of a file to our program.

ii. .org
The .org directive is used to indicate the beginning of address.

iii. LDI instruction
The LDI instruction copies the 8-bit data *K* into the general purpose registers Rd ($16 \leq d \leq 31$). It has the following format:
LDI Rd, K

iv. OUT instruction
The OUT instruction tells the CPU to store the GPR to the I/O register.
OUT A, Rr ; store register to I/O location ($0 \leq r \leq 31$), ($0 \leq A \leq 63$)

v. RJMP instruction
It is a two byte unconditional branch instruction to jump within -2048 to +2048 words of memory relative to the address of the current PC.

Procedure

5. Develop the program using AVR Studio, and build its hex file.
6. Implement the circuit as shown in figure 2.1 and give the power.
7. Burn the hex file into ATMega32 chip using eXtreme Burner software.
8. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
;EXP. NO. 13
;Aim: To send one byte of data on to PORTB of ATMEGA32
; File Name: 13_EXP_LED_ASM.asm
; Created: 08/02/2018 14:36:28
; Author :
; Register No :

 .include "M32DEF.inc"
 .ORG 0

;---------Setting Port B as output----------------
            LDI R16,0XFF
            OUT DDRB,R16
;-------sending 11011011 to port B
            LDI R16,0B11011011
            OUT PORTB,R16
L:          RJMP L              ; INFINITE LOOP
```

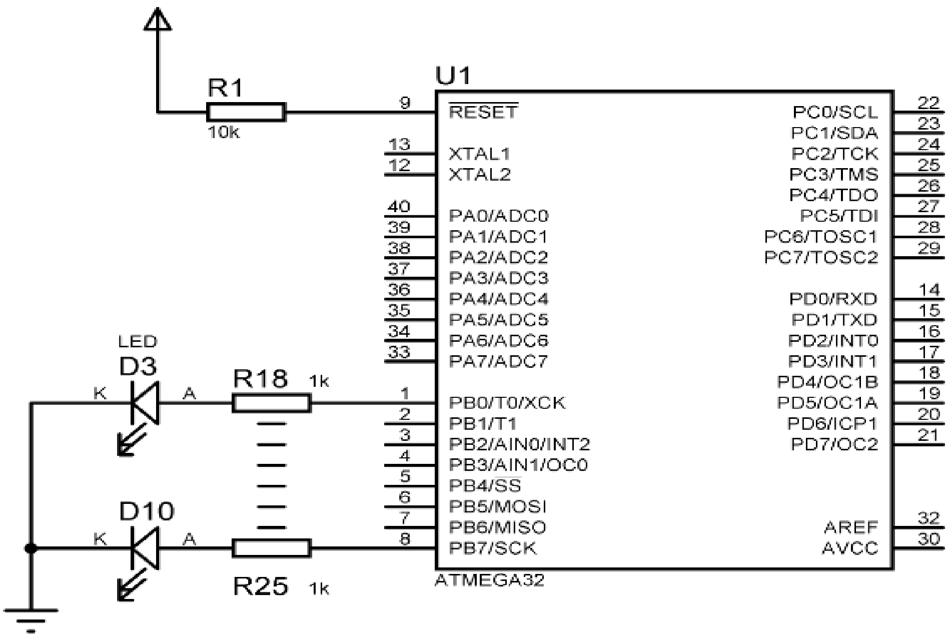


Figure 13.1 : Circuit Diagram

EX.NO. 14 DATE :

LED INTERFACING - 5

Aim: To send data from 00 to FF to PORTB of ATMEGA32 by giving sufficient delay between out operation. Write program in ASM language.

Theory

vi. CALL instruction
It calls a subroutine. When a subroutine is called, the processor first saves the address of the instruction just below the CALL instruction on the stack, and then transfers control to that subroutine.

vii. RET instruction
When the RET instruction at the end of the subroutine is executed, the top location of the stack is copied back to the PC and SP is incremented. i.e;, the address of the instruction below the CALL is loaded into the PC, and the instruction below the CALL is executed.

viii. BRNE instruction
The BRNE (branch if not equal) instruction uses the zero flag in the status register. If the zero flag is set, it jumps to the target location.

Procedure

9. Develop the program using AVR Studio, and build its hex file.
10. Implement the circuit as shown in figure 3.1 and give the power.
11. Burn the hex file into ATMega32 chip using eXtreme Burner software.
12. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
; 14_EXP_LED_ASM.asm
; Created: 08/02/2018 18:13:27
; Author : cts6
;sending 00 to FF to Port B

.include "M32DEF.inc"

 .ORG 0          ;       starting address of the code
;---------Initializing STACK POINTER with last SRAM location ----
            LDI R16, HIGH(RAMEND)
            OUT SPH, R16
            LDI R16, LOW(RAMEND)
            OUT SPL, R16
;---------Setting Port B as output-----------------
            LDI R16,0XFF
            OUT DDRB,R16

;-------sending 00 to ff to port B --------------
L:          INC R16
            OUT PORTB,R16
            CALL DELAY
            RJMP L

;-------------DELAY SUBROUTINE --------------------------
; --- Adjust the initial values of R17, R18, R19 to correct delay
.ORG 0X300
DELAY:      LDI R17, 0XF
AGAIN1:     LDI R18, 0XFF
AGAIN2:     LDI R19, 0XFF
AGAIN3:     NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DEC R19
            BRNE AGAIN3
            DEC R18
            BRNE AGAIN2
            DEC R17
            BRNE AGAIN1
            RET
```

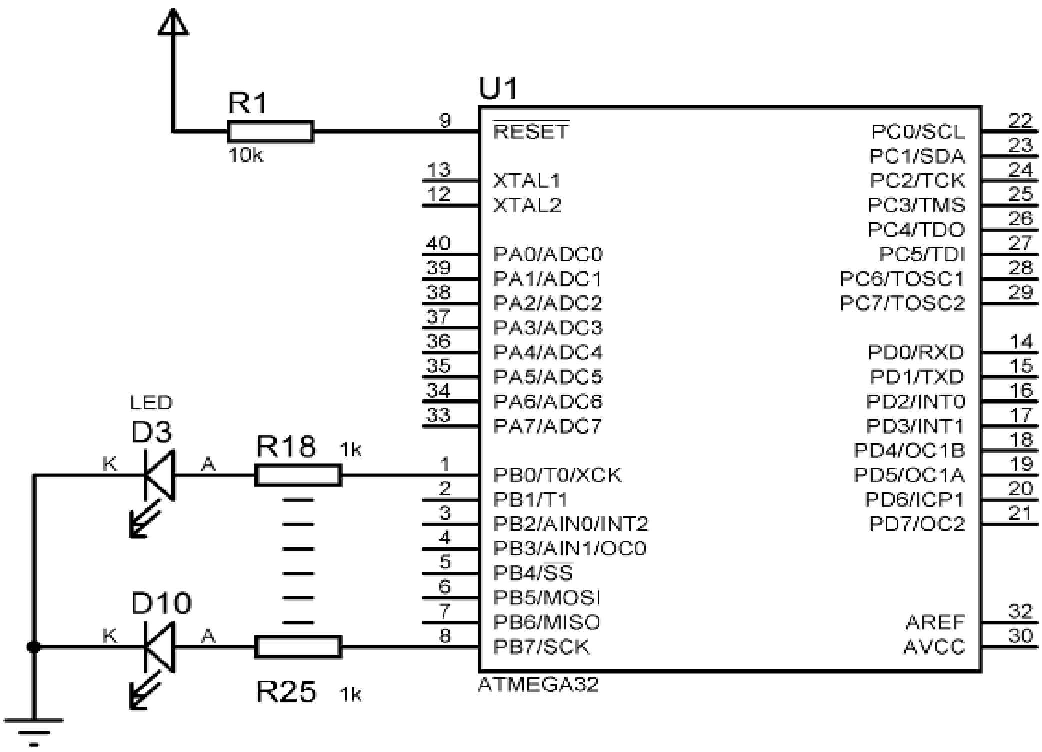


Figure 14.1 : Circuit Diagram

EX.NO. 15 DATE :

Port pins as input- Assembly

Aim: To read switches connected to PORTD and output the contents to LEDs connected to PORTB. Write program in ASM.

Theory

PINx register

PINx (Port IN) register is used to read data from port pins. In order to read the data from port pin, first you have to change port's data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins. There are two input modes. Either you can use port pins as tri stated inputs or you can activate internal pull up. In order to enable internal pull up, you have to write 1 to the corresponding PORTx register.

IN instruction

The IN instruction tells the CPU to load one byte from an I/O register to the GPR

OUT instruction

The OUT instruction tells the CPU to store one byte from GPR to the I/O .

Procedure

13. Develop the program using AVR Studio, and build its hex file.
14. Implement the circuit as shown in figure 3.1 and give the power.
15. Burn the hex file into ATMega32 chip using eXtreme Burner software.
16. Observe the result

Result

The circuit has been implemented, program has been developed and created hex code, the hex code is burned on the microcontroller, and observed the working the embedded system.

Program

```
;
; 15_EXP_INPUT_ASM.asm
;
; Created: 09/02/2018 16:06:08
; Author : cts6
;

;This program demonstrates the working of ports as input and
output
;Program reads switches connected to PORTD
;and output the values to PORTB

.include "M32DEF.inc"


 .ORG 0          ;       starting address of the code

;---------Initializing STACK POINTER with last SRAM location ----
                LDI R16, HIGH(RAMEND)
                OUT SPH, R16
                LDI R16, LOW(RAMEND)
                OUT SPL, R16

;---------Setting Port B as output-----------------
                LDI R16,0XFF
                OUT DDRB,R16

;---------Setting Port D as input-----------------
                OUT PORTD,R16 ; activating pull up register
                LDI R16,0X00
                OUT DDRD,R16

;-------INPUTTING AND OUTPUTTING --------------
L:
                IN R16, PIND     ; Load Data from Port D to R16
                COM R16                 ; complementing R16
                OUT PORTB,R16           ; sending R16 to output Port B
                CALL DELAY              ; Calling delay
                RJMP L

;-------------DELAY SUBROUTINE --------------------------
; --- Adjust the initial values of R17, R18, R19 to correct delay
.ORG 0X300
```

```
DELAY:    LDI R17, 0X1
AGAIN1:   LDI R18, 0X1
AGAIN2:   LDI R19, 0XF
AGAIN3:   NOP
          NOP
          NOP
          NOP
          NOP
          NOP
          DEC R19
          BRNE AGAIN3
          DEC R18
          BRNE AGAIN2
          DEC R17
          BRNE AGAIN1
          RET
```

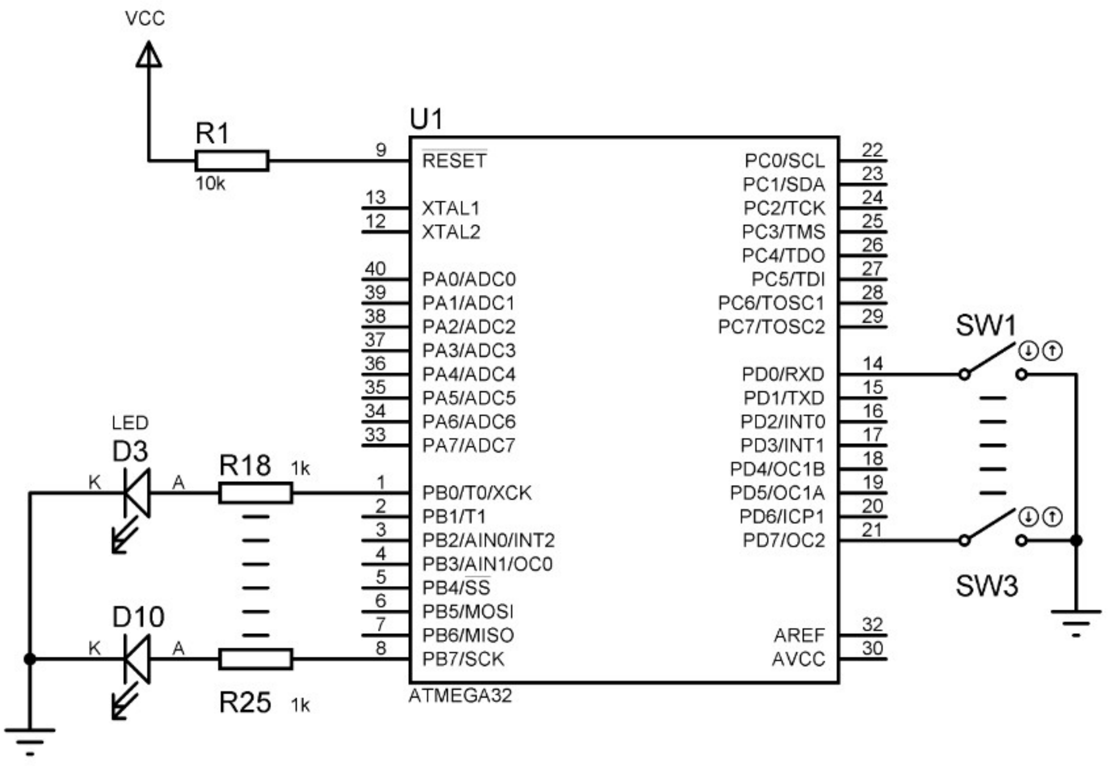


Figure 15.1