EE2016: Microprocessors Lab
Experiment # 3: Hardware Wiring and Programming for Interrupts by ASM and C-Programming using Atmel Atmega(8) AVR

Amizhthni PRK, EE21B015
Nachiket Dighe, EE21B093

October 8, 2022

Contents

List of Figures

1 Part 1

Interaction with peripherals in Atmel Atmega8 microcontroller.

1.1 Aim in this part 1 of experiment:

1 . Wire the microcontroller along with the given peripherals in a breadboard.

2 . Program the microcontroller to read the DIP switch values and display it in an LED using assembly programming. Basically blinking of an LED.

1.2 Equipments and Hardware Required

- Atmel AVR (Atmel8L) Chip - 1
- A breadboard with microprocessor socket
- 8-bit DIP switches
- 5 LEDs
- Capacitors, resistors and wires
- AVR Programmer (USB-ASP)
- A windows PC loaded with Atmel Studio 6.2 and AVR Burn-O-MAT (for burning ash)

1.3 Algortihm, Approach, Code Explanation

- This is explaining each line of code:
- .CSEG: Directive that denotes the start of a code segment.
- Note that DDRB and PINx and PORTx are all registers too
- We are setting all bits to 1(high) to make the port as output
- We can also give this as LDI R16, 0xFF.
- Data direction register for port B, it makes the DDRB i/o space to high
- We are setting all bits to 0(low) to make the port as input
- Data direction register for port D, it makes the DDRD i/o space to low
- Again forms a loop here
- This is done to turn off the LED
- Generally PORT is used with the port that has outputs, out command with port
- Now the port is off
- Generally PIN is used with the port that has inputs, in command with pin.
- When the input is given from a push button, it excites pind register which transfers the high bit to R16

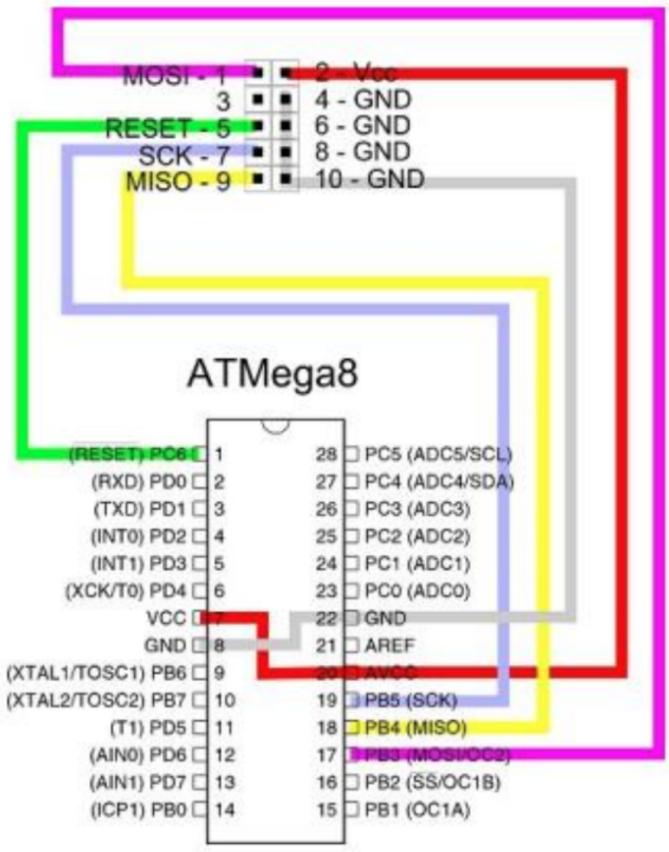

Figure 1: Microcontroller to In-System Programmer Connections

Figure 1: Connections overall have been made using this outline

Instruction	Description	Operation
ADD Rd, Rr	Unsigned Add without Carry	Rd←Rd + Rr
MUL Rd, Rr	Unsigned multiply	R1: R0 ← Rd * Rr
AND Rd, Rr	Bitwise logical AND	Rd ← Rd . Rr
OR Rd, Rr	Bitwise logical OR	Rd ← Rd | Rr
LDI Rd, K	Load Immediate a constant	Rd ← K
LSL Rd	Shift value to left by 1 bit	Rd ← Rd << 1
LSR Rd	Shift value to right by 1 bit	Rd ← Rd >>1
INC Rd	Increment by 1	Rd ← Rd + 1
Dec Rd	Decrement by 1	Rd ← Rd - 1

Figure 2: The exhaustive set of commands used

- This is essential to make the LED glow when the push button is pressed.
- Without this the led goes off on pressing the button.
- Logical and with immediate(logical between an immediate value and a constant
- This is used to denote the final stage of led, if R16 is high it glows else it goes off
- Loop jumps to the line containing again. Loop continues until input in push button is given.

1.4 Code

The code used for finding the sum of two 8-bit numbers is given below in listing 1.

```
.CSEG   //directive that denotes the start of a code segment.
              //Note that DDRB and PINx and PORTx are all registers too

LDI R16, 0x01  //We are setting all bits to 1(high) to make the port as
 ↪  output
                            //we can also give this as  LDI R16, 0xFF.

OUT DDRB, R16   //Data direction register for port B, it makes the DDRB i/o
 ↪  space to high

LDI R16, 0x00   //We are setting all bits to 0(low) to make the port as
 ↪  input

OUT DDRD, R16    //Data direction register for port D, it makes the DDRD
 ↪  i/o space to low

again: LDI R16, 0x00   //again forms a loop here
                                          //this is done to turn off the
 ↪  LED
       OUT PORTB, R16    //generally PORT is used with the port that has
 ↪  outputs, out command with port
                                          //now the port is off

       IN R16,PIND    //generally PIN is used with the port that has
 ↪  inputs, in command with pin.
                                //When the input is given from a push
 ↪  button, it excites pind register which transfers
                                //the high bit to R16

       COM R16  //This is essential to make the LED glow when the push
 ↪  button is pressed.
                  //Without this the led goes off on pressing the button.

       ANDI R16, 0x01//logical and with immediate(logical between an
 ↪  immediate value and a constant

       OUT PORTB, R16   //This is used to denote the final stage of led, if
 ↪  R16 is high it glows else it goes off
```

```
28
29          rjmp again  //loop jumps to the line containing again. Loop
    ↪  continues until input in push button is given.
```

Listing 1: Code for blinking an LED

1.5 Circuit Schematic

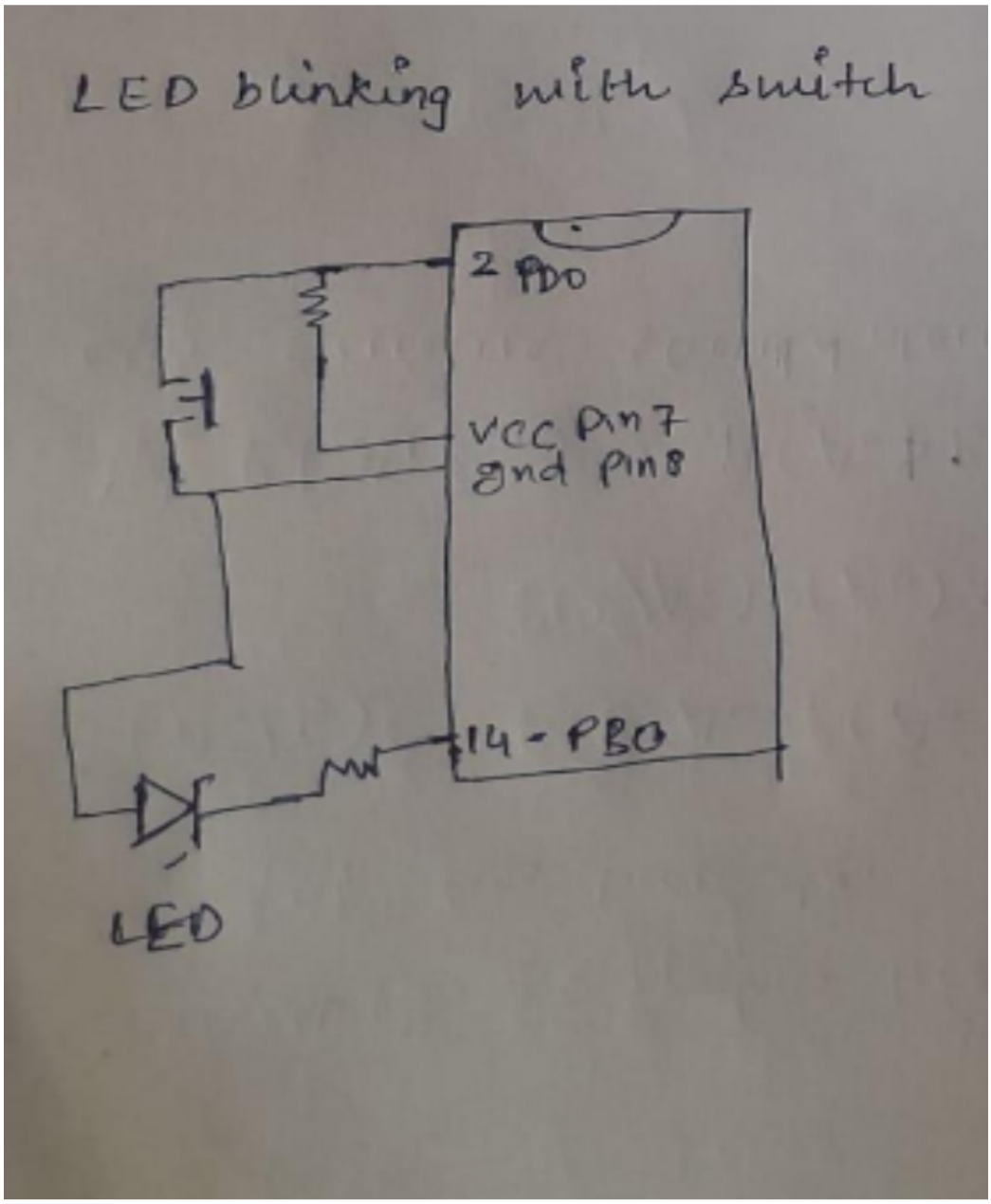

Figure 3: Schematic for blinking LEDs

1.6 Blinking LED outputs

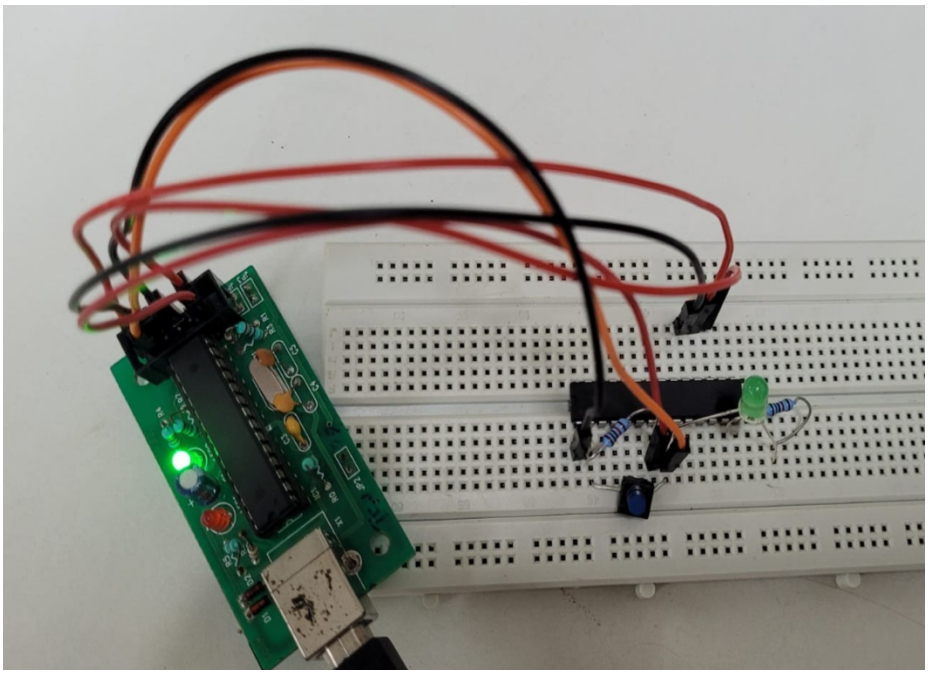

Figure 4: When no input is given, led does not blink

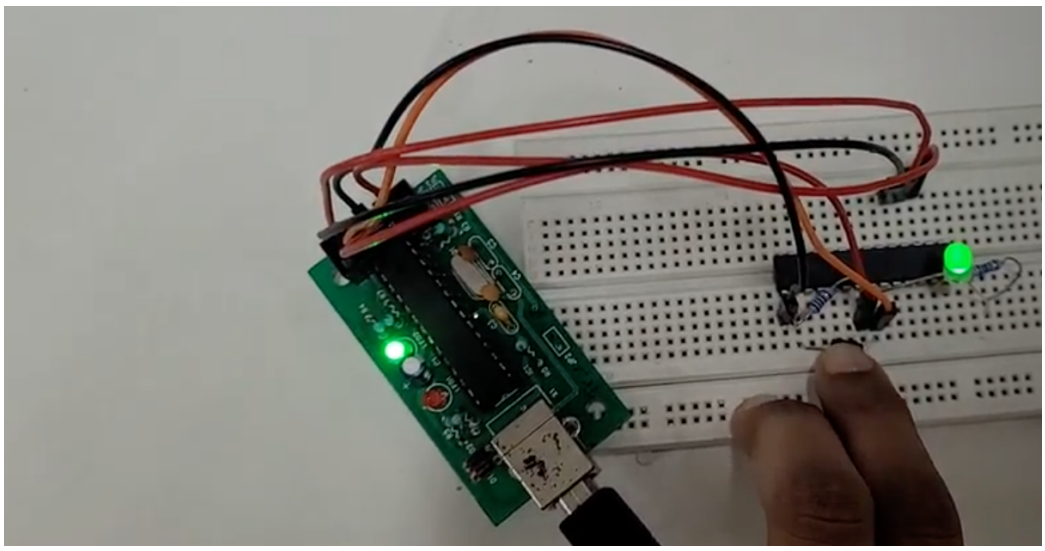

Figure 5: When the input is given, led blinks

2 Part 2

4 bit addition of two unsigned nibbles from the 8 bit dip input switch and display the result obtained in LEDs.

2.1 Aim in this part 2 of experiment:

1 Program the microcontroller to perform the addition of two four bit numbers which are read from the DIP switches connected to a port and display the result using LED's connected to another port.

2.2 Algortihm, Approach, Code Explanation

- The two 4-bit numbers are inputted into an 8-bit register.
- The number is split into two halves and added. The output is stored in another register.
- The output is taken at port C.
- Here the input is given at port D and output is obtained at port C.
- In the example, we add 1101(13) and 0111(7) to get 10100(20). **Flowchart**:
- START
- Use PIND AS INPUT
- Use PORTC AS OUTPUT
- TAKE INPUT FROM PIND
- Store numbers in R21, R20
- Make two nibbles in R21, R20
- Add R21 and R20
- Print output at port C
- END

2.3 Code

The code used for finding the sum of two 4-bit numbers and flashing it on LEDs is given below in listing 2.

```
1 /*
2  *
3  *        Addition of two unsigned nibbles taken from a DIP switch
4  *
5  *        INPUT - from DIP switch connected to PORTD
6  *        OUTPUT - To the LEDs connected to PORTC
7  *
8  */
9
```

```
#include "m8def.inc"

START:
        LDI R16, 0x00; // Setting all bits to zero to make the port input
        OUT DDRD, R16;  Setting PORTD to INPUT

        LDI R16, 0xFF; // Setting all bits to one to make the port as
↪   output
        OUT DDRC, R16;  Setting PORTC to OUTPUT

ADDITION:
        IN R21, PIND;  // R21 <-- (<NUM2><NUM1>)
        MOV R20, R21;  // Making copy of R21 in R20 for having the 2
↪   numbers in separate registers
        ANDI R20, 0xF0; // Assigning R20 as "<NUM2>0000"
        SWAP R20;  // Swapping higher and lower nibbles of R20. R20 <--
↪   "0000<NUM2>"
        ANDI R21, 0x0F; // Assigning R21 as "0000<NUM1>"
        ADD R20, R21;  //R20 <-- R20 + R21

END:
    OUT PORTC, R20;         //PORTC <-- R20
        NOP; //End of program
```

Listing 2: Addition and LED blink

2.4 Circuit Schematic

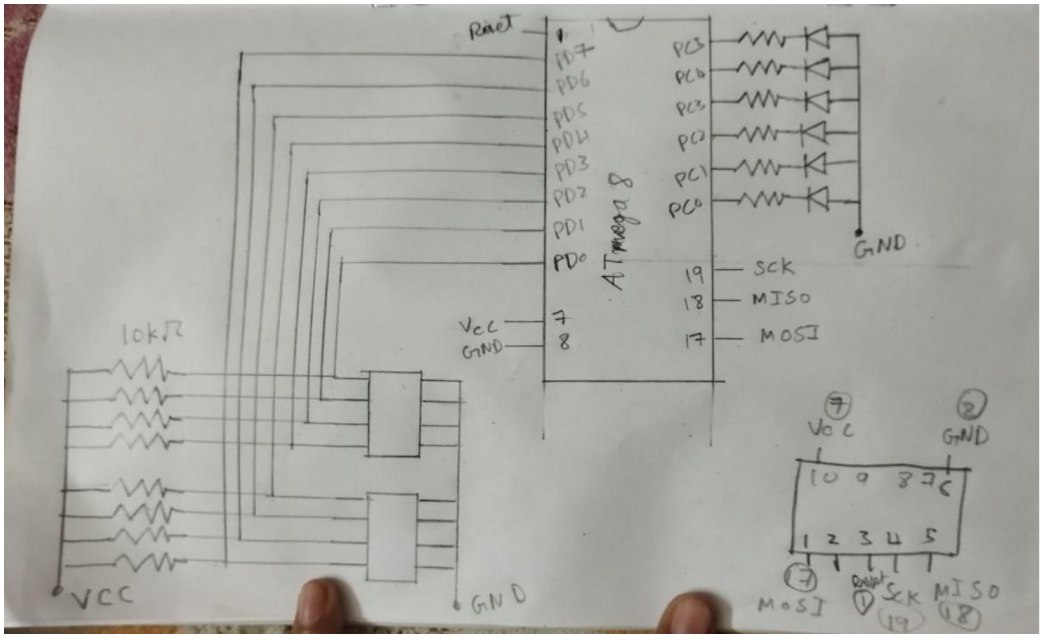

Figure 6: Circuit schematic for addition

2.5 Addition output on LEDs

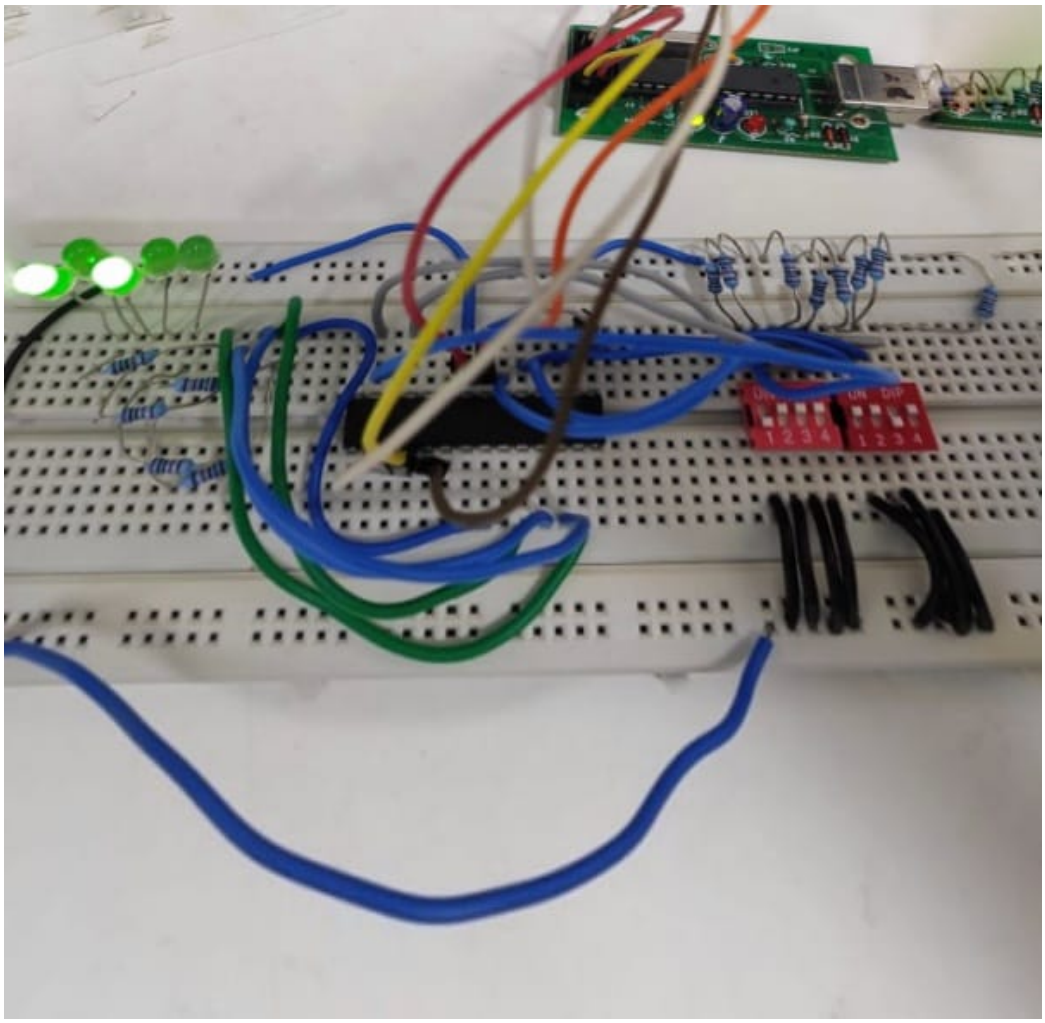

Figure 7: Output for 1101 + 0111 = 10100

3 Part 3

4 bit multiplication of two unsigned nibbles and display the result in LEDs.

3.1 Aim in this part 3 of experiment:

1 Program the microcontroller to perform the multiplication of two four bit numbers which are read from the DIP switches connected to a port and display the result using LED's connected to another port.

3.2 Algorithm, Approach, Code Explanation

- The two 4-bit numbers are inputted into an 8-bit register.
- The number is split into two halves and multiplied and the output is stored in another register.
- The output is taken at port C.
- Here the input is given at port D and output is obtained at port C.
- The particular example has the numbers 1100(Decimal: 12) and 1(Decimal: 15) as input.
- The output is clearly visible as 10110100(Decimal: 180) which is the desired output.
- Hence the multiplication has been carried out successfully.

3.3 Flowchart

- START
- Declare PIND as input
- Declare portc as output
- Take input from pind
- Make copy in register R22 and R21
- Make two nibbles in R22 and R21
- Multiply R22 and R21
- Move result to R20
- Print output at portC
- END

3.4 Code

The code used for finding the prodcut of two 4-bit numbers and flashing it on LEDs is given below in listing 3.

```
/*
 *
 *        Multiplication of two unsigned nibbles taken from a DIP switch
 *
 *        INPUT - from DIP switch connected to PORTD
 *        OUTPUT - To the LEDs connected to PORTC
 *
 */

#include "m8def.inc"

START:
    LDI R16, 0x00; //Setting all bits to zero to make the portD as input
    OUT DDRD, R16;  //Setting PORTD to INPUT

    LDI R16, 0xFF; //Setting all bits to one to make the portC as output
    OUT DDRC, R16;  //Setting PORTC to OUTPUT

        LDI R16, 0x00;        //clearing productL register
        LDI R17, 0x00;        //clearing productH register
        LDI R18, 0x00;        //clearing temporary register

INPUT:
    IN R21, PIND;   //R21 <-- (<NUM2><NUM1>)
    MOV R20, R21;   //Making copy of R21 in R20 for having the 2 numbers in
    ↪  separate registers
    ANDI R20, 0xF0; //Assigning R20 as "<NUM2>0000"
    SWAP R20;  //Swapping higher and lower nibbles of R20. R20 <--
    ↪  "0000<NUM2>"
    ANDI R21, 0x0F;  //Assigning R21 as "0000<NUM1>"

MULTIPLY1:
        CLC;   //clear Carry Bit
        ROR R21;   //right rotation of Register21
        BRCC MULTIPLY2;   //go to MULTIPLY2 label when last bit (carry now)
    ↪  is cleared.
        ADD R16, R20;   //add contents of R20 to R16
        ADC R17, R18;   //add contents of R18 to R17

MULTIPLY2:
        CLC; //clear Carry bit
        ROL R20;  //left rotation of Register20
        ROL R18; //left rotation of register18
        TST R21;
        BRNE MULTIPLY1;   //go to MULTIPLY1 label when zero flag is set to
    ↪  1.

END:
```

```
45          OUT PORTC, R16;    //Assigning only low byte of output to PORTC due
    ↪   to space constraint
46          NOP;    //End of Program
```

Listing 3: Multiplication and LED blink

3.5 Circuit Schematic

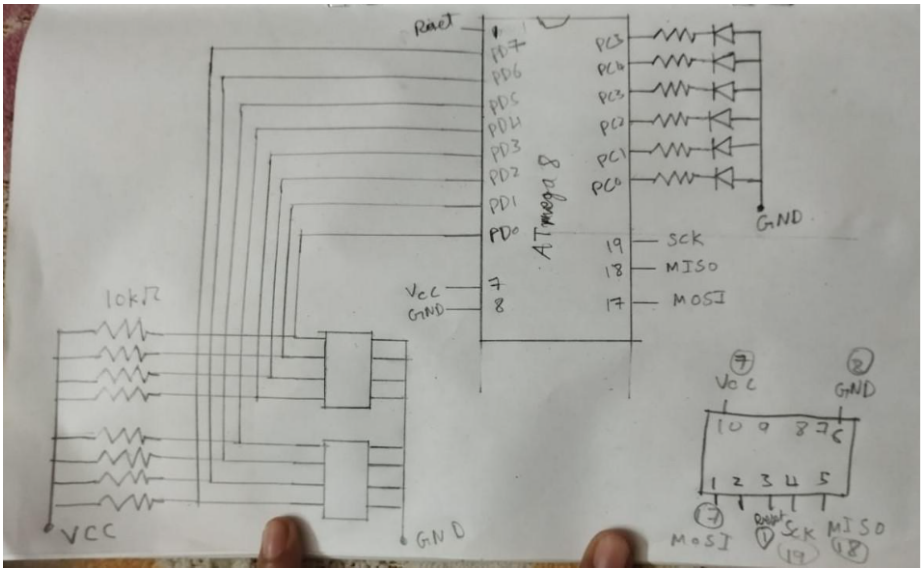

Figure 8: Circuit schematic for multiplication

3.6 Multiplication output on LEDs

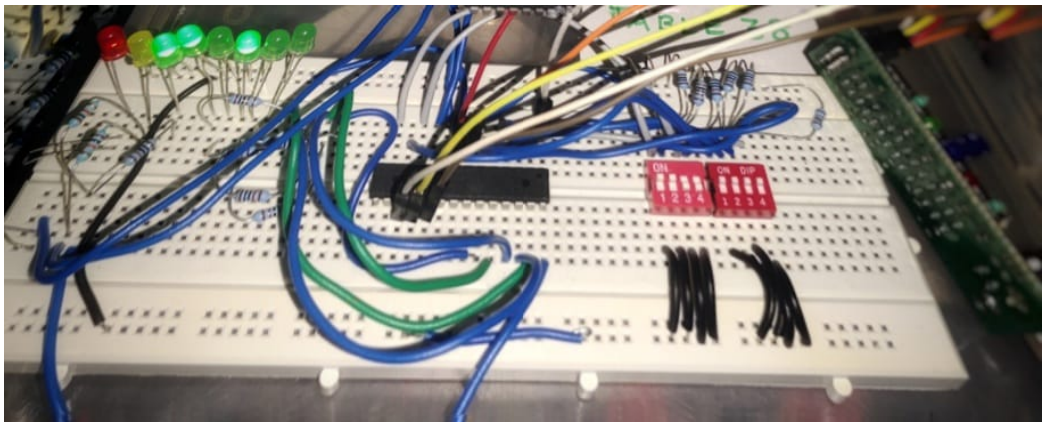

Figure 9: Output for 1100*1111 = 10110100