EE2016 - MICROPROCESSORS

EXP -3 LAB REPORT

EE21B098 - SAILENDRA

EE21B016 - ANANYA DAS

EXPERIMENT 3: Hardware Wiring and Programming for demonstrating Blinking LED, Addition and Multiplication using Atmel Atmega (8) AVR

AIM:

This experiment introduces assembly programming and interaction with peripherals in Atmel Atmega8 microcontroller.

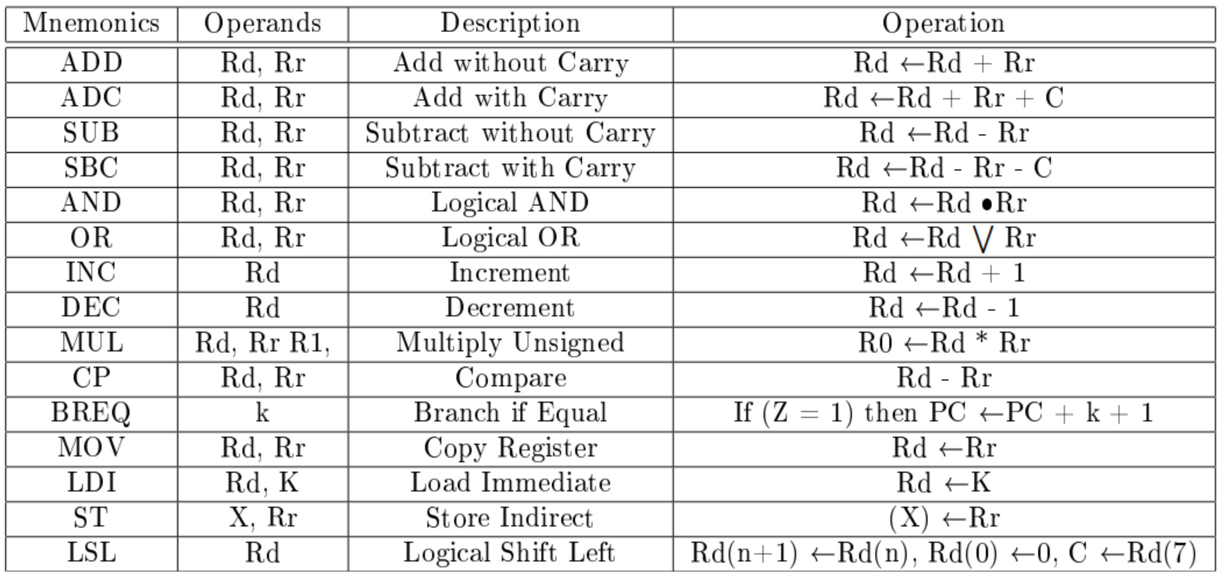
1. Wire the microcontroller along with the given peripherals in a breadboard.
2. Program the microcontroller to read the DIP switch values and display it in an LED using assembly programming.
3. Program the microcontroller to perform the addition and multiplication of two four-bit numbers which are read from the DIP switches connected to a port and display the result using LEDs connected to another port.

Equipment (Hardware/ Software) Required:

"To perform this experiment, the following components are required."
1. Atmel AVR (Atmel8L) Chip - I

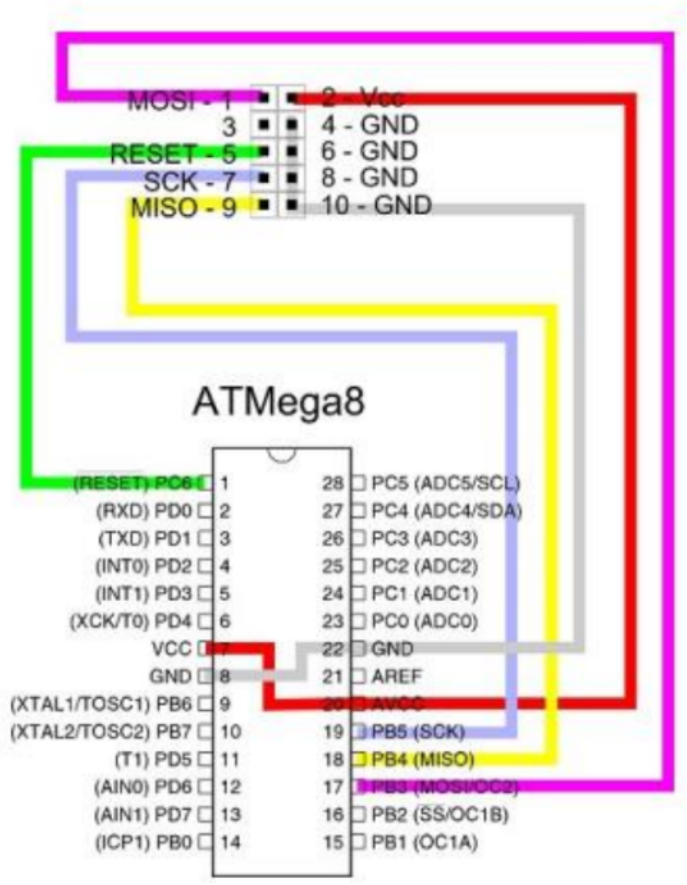
2. A breadboard with microprocessor socket
3. 8-bit DIP switches 4. 5 LEDs
"5. Capacitors, resistors and wires"
6. AVR Programmer (USB-ASP)
7. A windows PC loaded with Atmel Studio 6.2 and AVR Burn-0-MAT (for burning _ash)

Set of instructions used in the experiment:

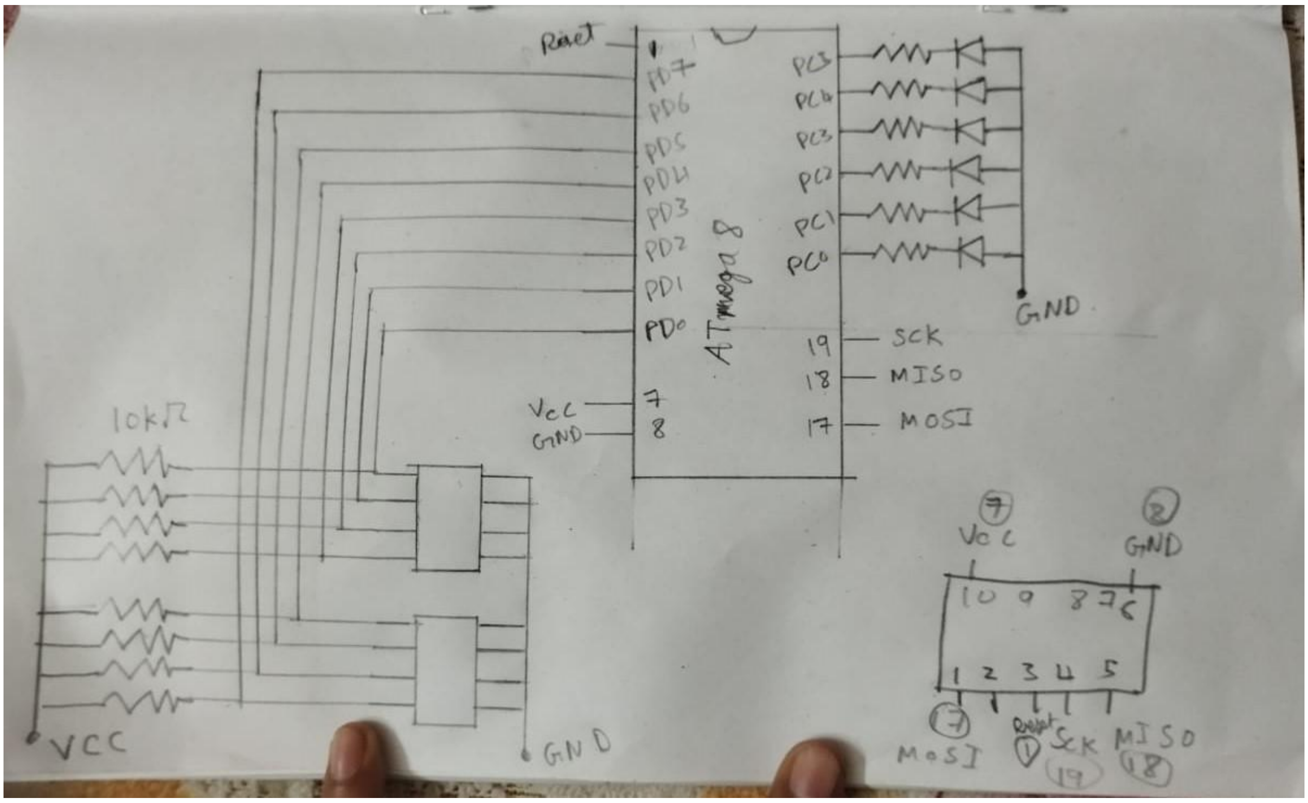


Mnemonics	Operands	Description	Operation
ADD	Rd, Rr	Add without Carry	Rd ←Rd + Rr
ADC	Rd, Rr	Add with Carry	Rd ←Rd + Rr + C
SUB	Rd, Rr	Subtract without Carry	Rd ←Rd - Rr
SBC	Rd, Rr	Subtract with Carry	Rd ←Rd - Rr - C
AND	Rd, Rr	Logical AND	Rd ←Rd •Rr
OR	Rd, Rr	Logical OR	Rd ←Rd ⋁ Rr
INC	Rd	Increment	Rd ←Rd + 1
DEC	Rd	Decrement	Rd ←Rd - 1
MUL	Rd, Rr R1,	Multiply Unsigned	R0 ←Rd * Rr
CP	Rd, Rr	Compare	Rd - Rr
BREQ	k	Branch if Equal	If (Z = 1) then PC ←PC + k + 1
MOV	Rd, Rr	Copy Register	Rd ←Rr
LDI	Rd, K	Load Immediate	Rd ←K
ST	X, Rr	Store Indirect	(X) ←Rr
LSL	Rd	Logical Shift Left	Rd(n+1) ←Rd(n), Rd(0) ←0, C ←Rd(7)

Basic connections:



CONNECTIONS FOR ARITHMETIC OPERATIONS:



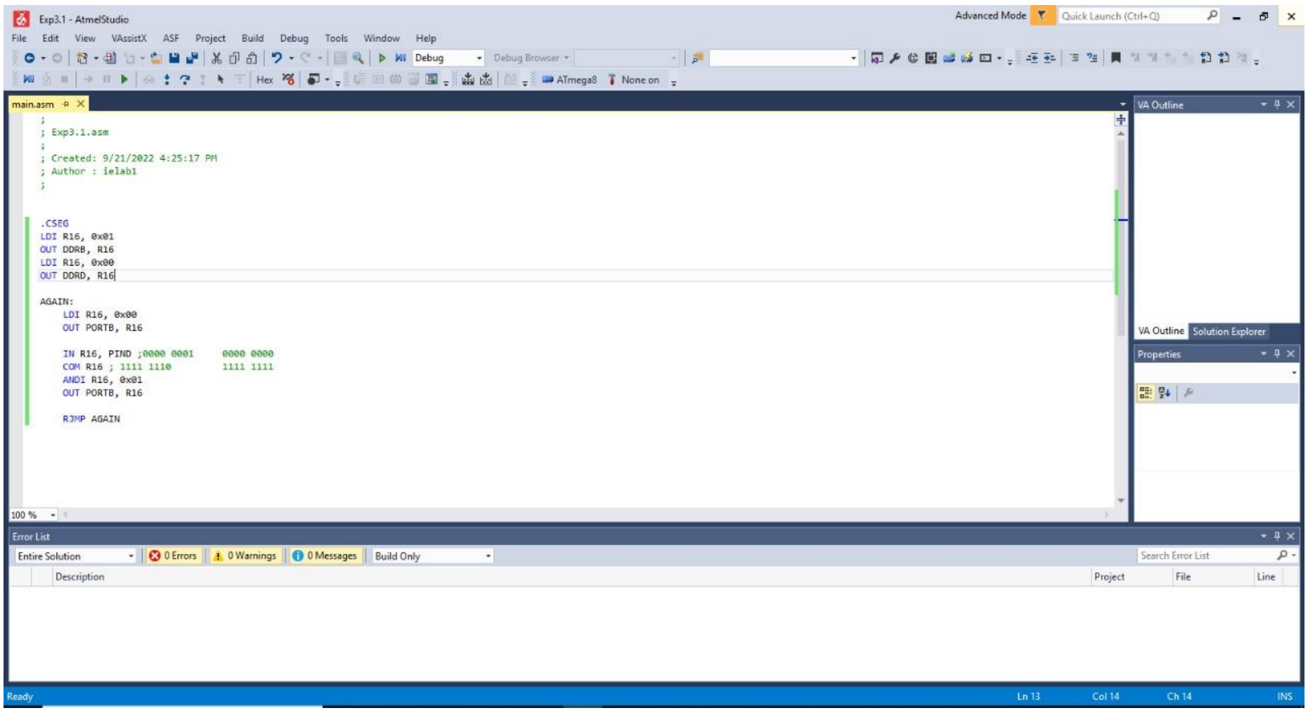
1) Blinking LED

To control an LED using a DIP switch.

1.1)CODE

```
.CSEG
LDI R16, 0x01
OUT DDRB, R16
LDI R16, 0x00
OUT DDRD, R16
again: LDI R16, 0x00
       OUT PORTB, R16
       IN R16,PIND
       COM R16
       ANDI R16, 0x01
       OUT PORTB, R16
       rjmp again
```

1.2)Code execution

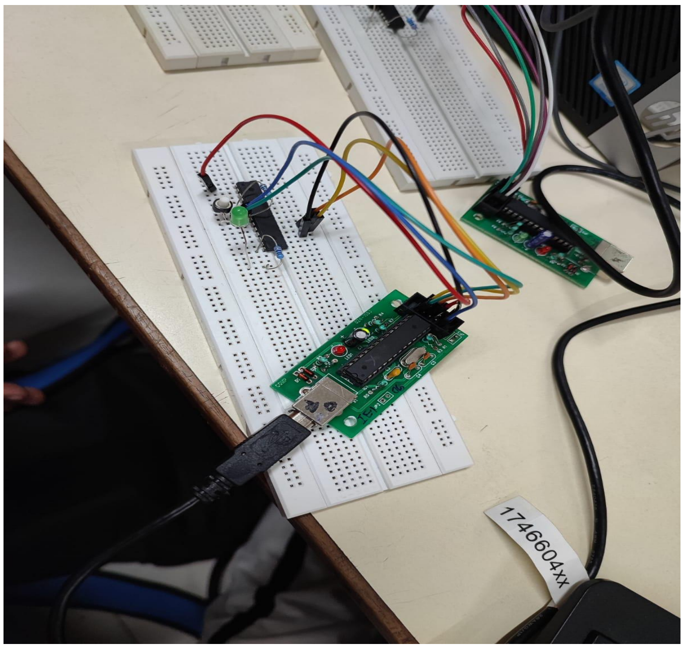


```
;
; Exp3.1.asm
;
; Created: 9/21/2022 4:25:17 PM
; Author : ielab1
;


.CSEG
LDI R16, 0x01
OUT DDRB, R16
LDI R16, 0x00
OUT DDRD, R16

AGAIN:
    LDI R16, 0x00
    OUT PORTB, R16

    IN R16, PIND ;0000 0001     0000 0000
    COM R16 ; 1111 1110         1111 1111
    ANDI R16, 0x01
    OUT PORTB, R16

    RJMP AGAIN
```

1.3)Explanation

1. The circuit consists of an LED which is connected via a DIP switch to Atmega8.
2. We have used the DIP switch to control the voltage given to LED hence turning it on or off.
3. Output is taken at PIN 14 of the IC.

1.4)Circuit/Output



1.5) Inference :

Here, as seen in the circuit, the LED is connected to DIP switch to IC and is grounded. On pressing the switch, the desired output was obtained. The LED was controlled using DIP switch.

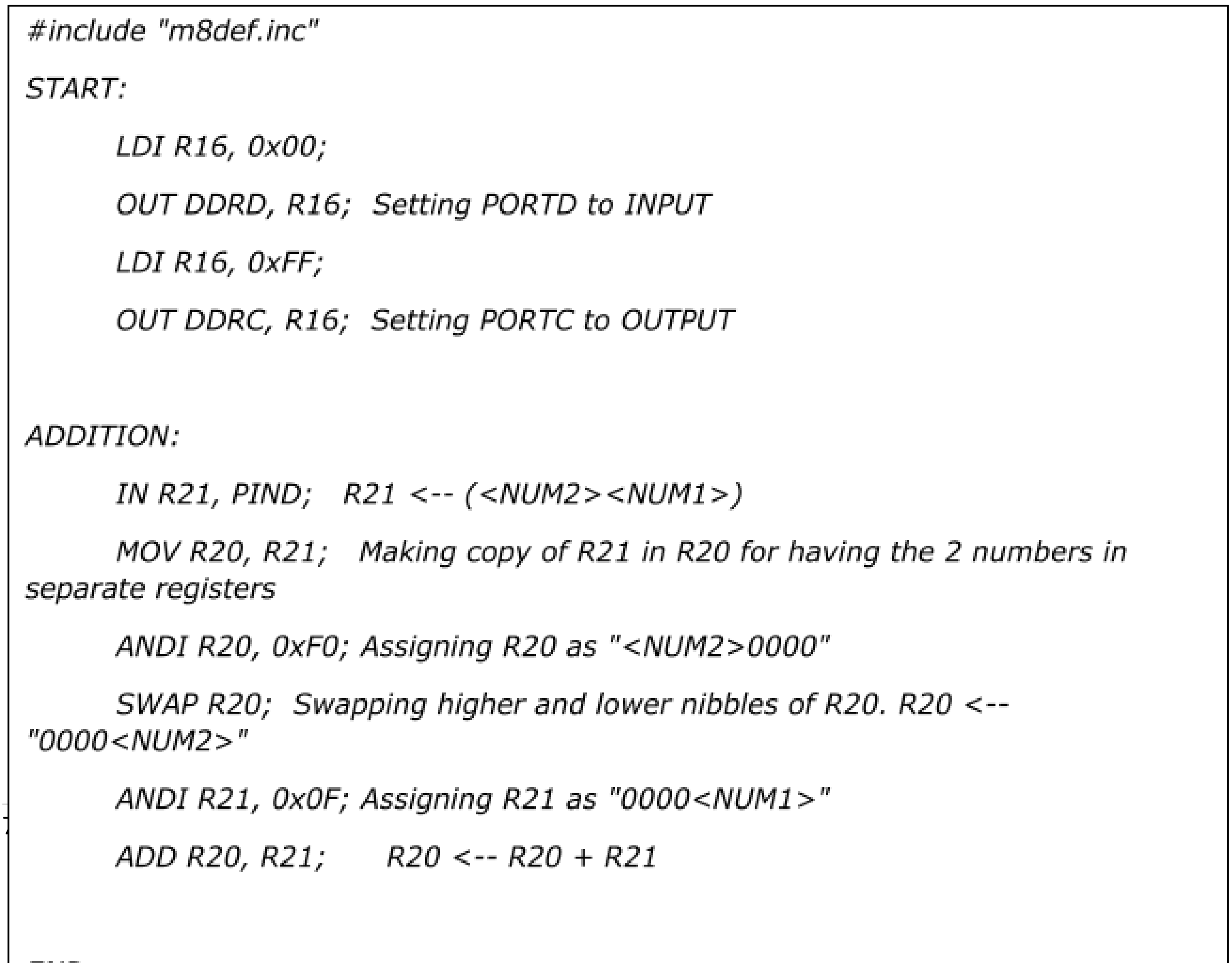
1.6)Videos and other outputs :

Link:
https://drive.google.com/file/d/1P0EuitrMjohCaejC_0GPh4AlncC6Y9PM/view?usp=sharing

2) Addition operation using Atmega8

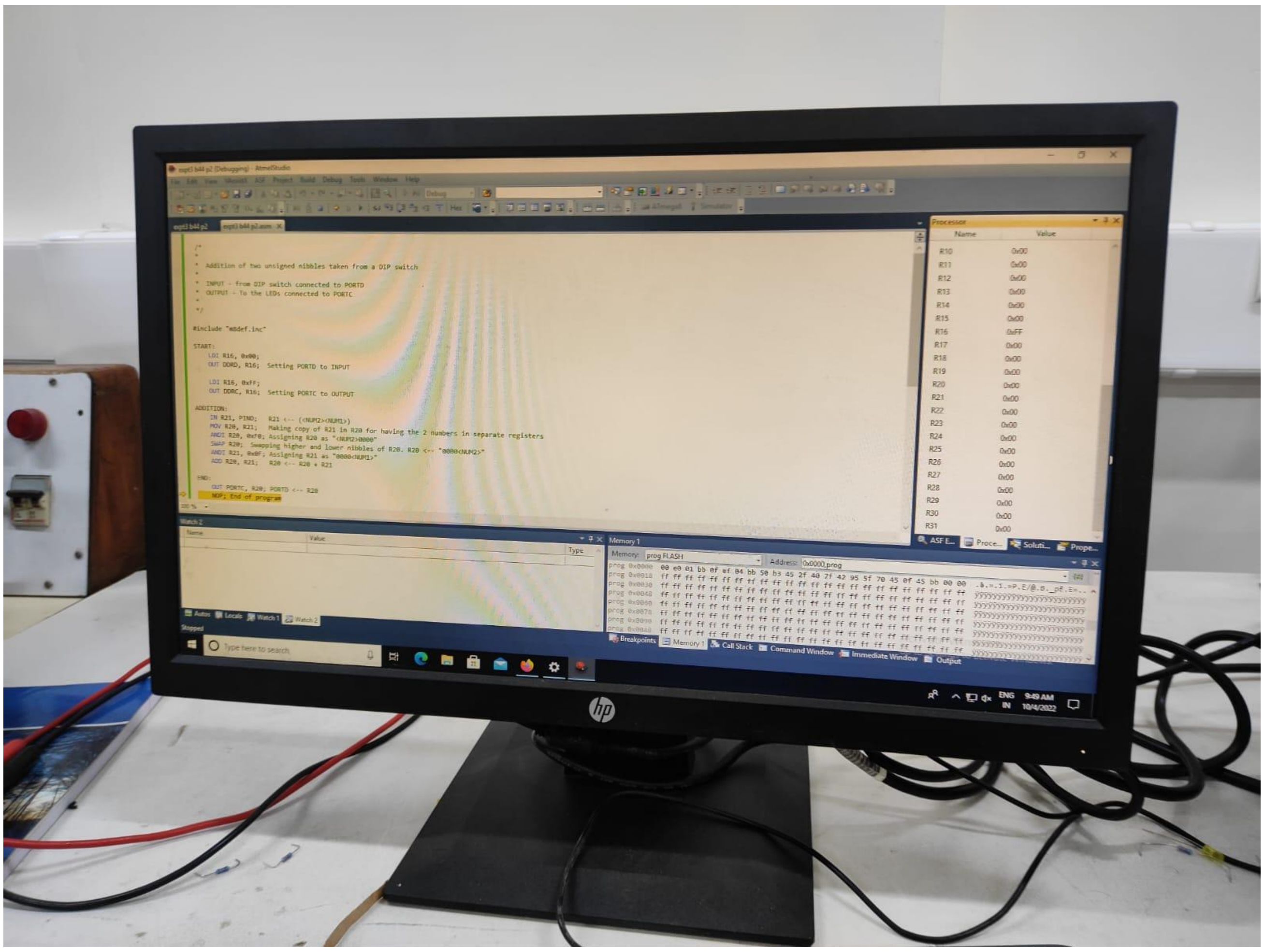
4-bit addition of two unsigned nibbles from the 8-bit dip input switch and display the result obtained in LEDs.

2.1) CODE



```
#include "m8def.inc"

START:

        LDI R16, 0x00;

        OUT DDRD, R16;  Setting PORTD to INPUT

        LDI R16, 0xFF;

        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
```

2.2) Code Execution



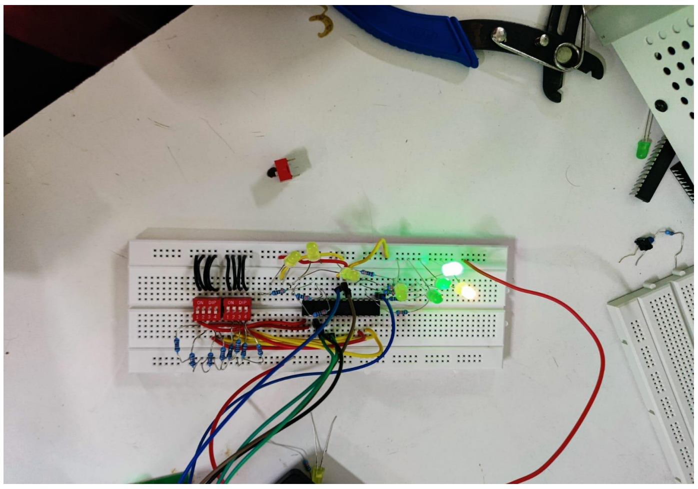
```
/*
 *
 *  Addition of two unsigned nibbles taken from a DIP switch
 *
 *  INPUT - from DIP switch connected to PORTD
 *  OUTPUT - To the LEDs connected to PORTC
 *
 */

#include "m8def.inc"

START:
    LDI R16, 0x00;
    OUT DDRD, R16;  Setting PORTD to INPUT

    LDI R16, 0xFF;
    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; PORTD <-- R20
    NOP; End of program
```

Name	Value
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0xFF
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00
R26	0x00
R27	0x00
R28	0x00
R29	0x00
R30	0x00
R31	0x00

2.3) Explanation

The two 4-bit numbers are inputted into an 8-bit register. The number is split into two halves and added and the output is stored in another register. The output is taken at port C.

2.4) Circuit/Output



2.5) Inference

Here we have given the inputs at port D and output is obtained at point C . We have given 0 and 256(FF)as the inputs and got the 256 as output at register R16.

AVR Burn-o-MAT : Successfully uploaded the code into IC

3) Multiplication operation using Atmega8

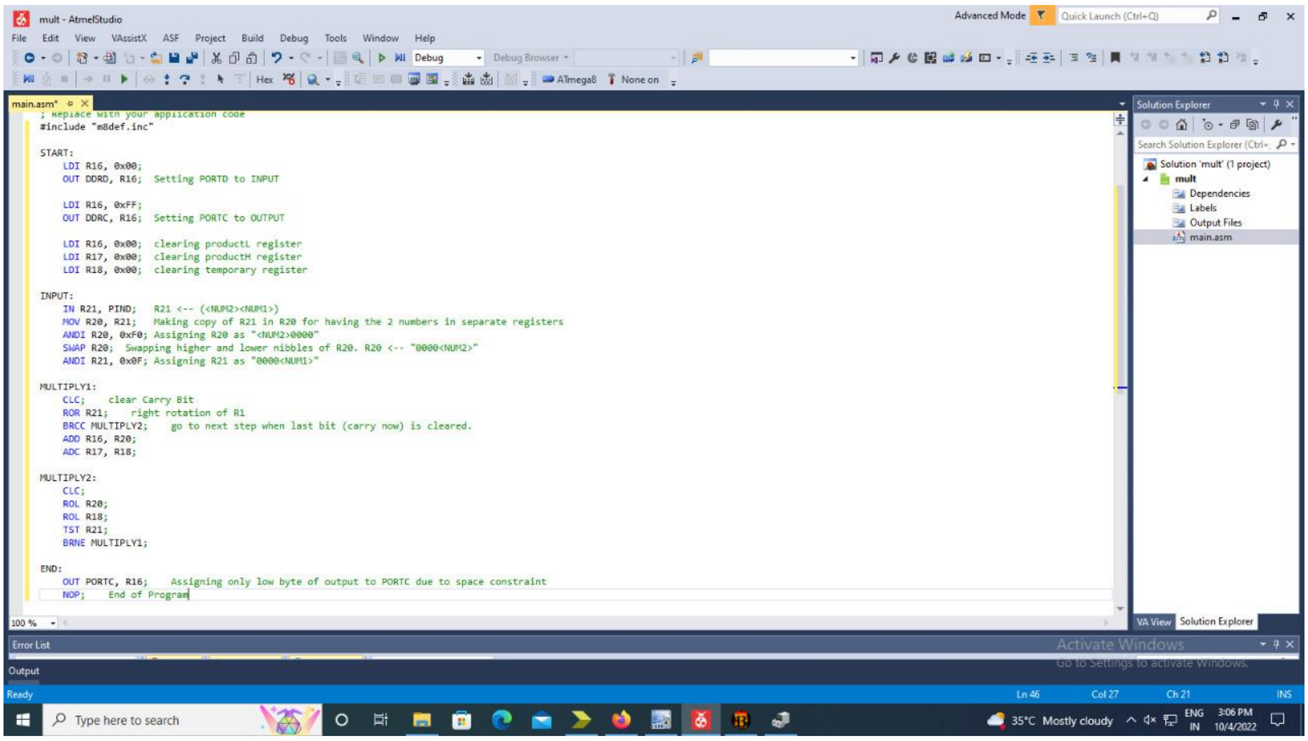
4-bit multiplication of two unsigned nibbles and display the result in LEDs.

3.1) CODE

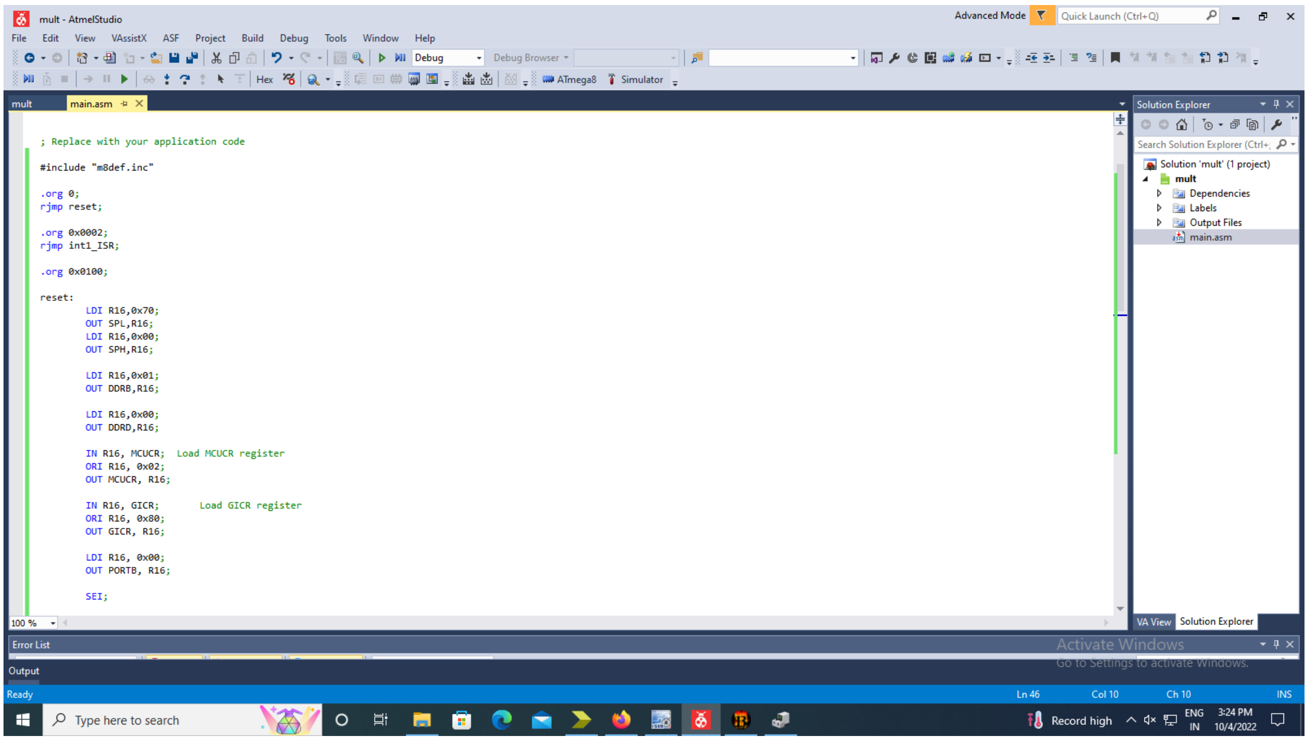
```
*


#include "m8def.inc"


START:
   LDI R16, 0x00;
   OUT DDRD, R16;  Setting PORTD to INPUT


   LDI R16, 0xFF;
   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 R1
```

```
	BRCC MULTIPLY2;		go to next step when last bit (carry now) is cleared.

	ADD R16, R20;

	ADC R17, R18;


MULTIPLY2:

	CLC;

	ROL R20;

	ROL R18;

	TST R21;

	BRNE MULTIPLY1;


END:

	OUT PORTC, R16;	Assigning only low byte of output to PORTC due to space constraint

	NOP;	End of Program
```

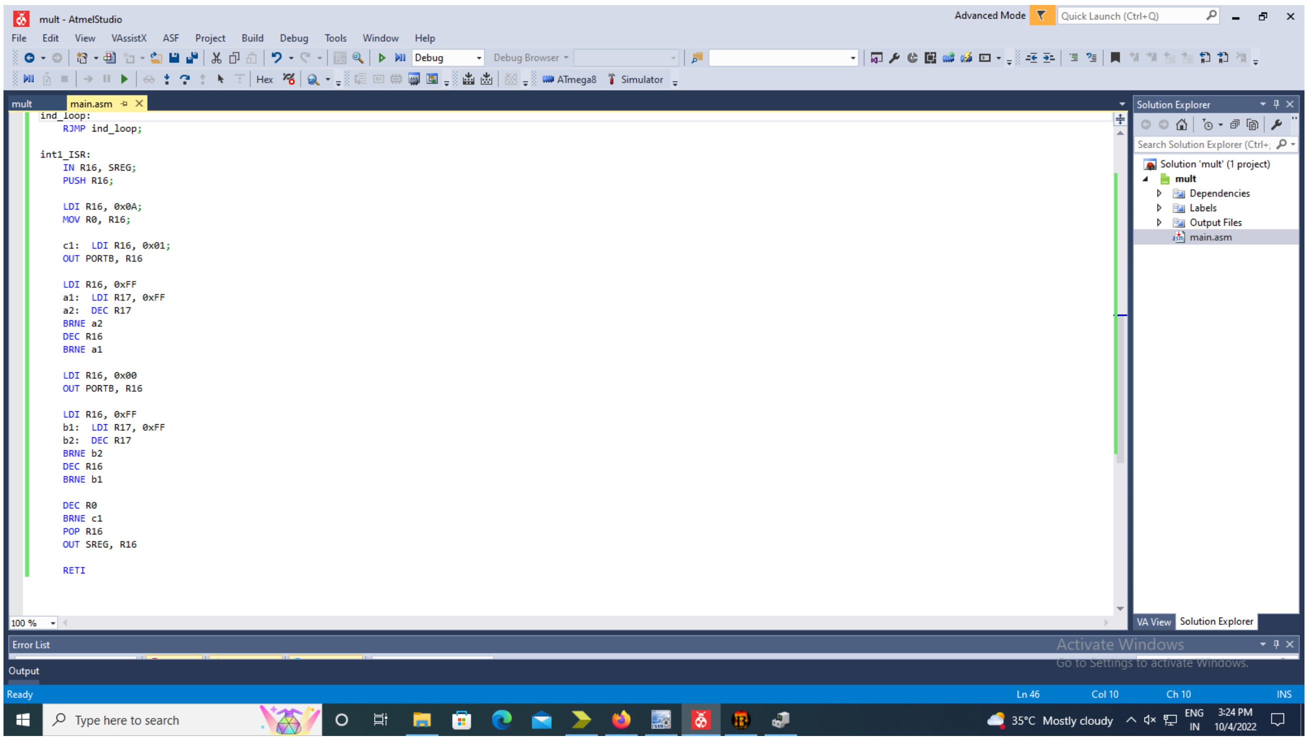
3.2) Code Execution



```
; Replace with your application code
#include "m8def.inc"

START:
    LDI R16, 0x00;
    OUT DDRD, R16;  Setting PORTD to INPUT

    LDI R16, 0xFF;
    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 R1
    BRCC MULTIPLY2;    go to next step when last bit (carry now) is cleared.
    ADD R16, R20;
    ADC R17, R18;

MULTIPLY2:
    CLC;
    ROL R20;
    ROL R18;
    TST R21;
    BRNE MULTIPLY1;

END:
    OUT PORTC, R16;    Assigning only low byte of output to PORTC due to space constraint
    NOP;    End of Program
```



```
; Replace with your application code

#include "m8def.inc"

.org 0;
rjmp reset;

.org 0x0002;
rjmp int1_ISR;

.org 0x0100;

reset:
        LDI R16,0x70;
        OUT SPL,R16;
        LDI R16,0x00;
        OUT SPH,R16;

        LDI R16,0x01;
        OUT DDRB,R16;

        LDI R16,0x00;
        OUT DDRD,R16;

        IN R16, MCUCR;  Load MCUCR register
        ORI R16, 0x02;
        OUT MCUCR, R16;

        IN R16, GICR;       Load GICR register
        ORI R16, 0x80;
        OUT GICR, R16;

        LDI R16, 0x00;
        OUT PORTB, R16;

        SEI;
```

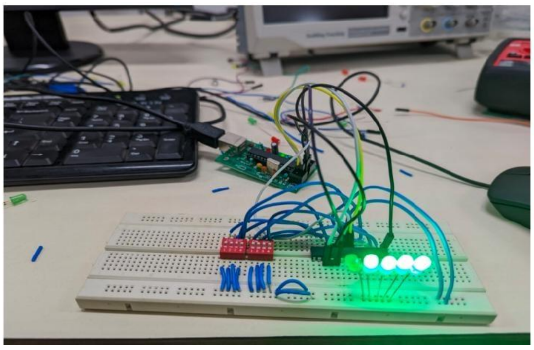


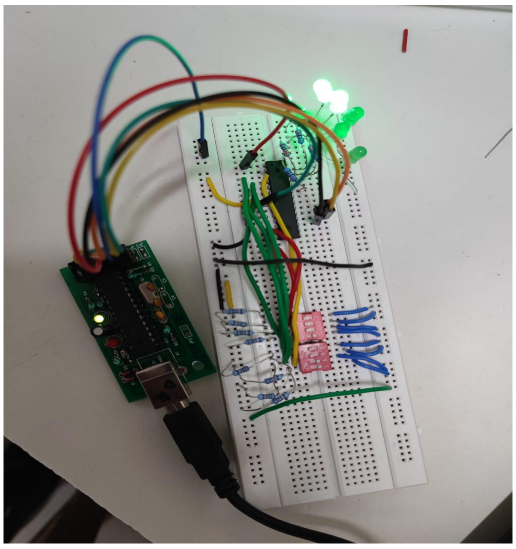
```
ind_loop:
    RJMP ind_loop;

int1_ISR:
    IN R16, SREG;
    PUSH R16;

    LDI R16, 0x0A;
    MOV R0, R16;

    c1:  LDI R16, 0x01;
    OUT PORTB, R16

    LDI R16, 0xFF
    a1:  LDI R17, 0xFF
    a2:  DEC R17
    BRNE a2
    DEC R16
    BRNE a1

    LDI R16, 0x00
    OUT PORTB, R16

    LDI R16, 0xFF
    b1:  LDI R17, 0xFF
    b2:  DEC R17
    BRNE b2
    DEC R16
    BRNE b1

    DEC R0
    BRNE c1
    POP R16
    OUT SREG, R16

    RETI
```

3.3) 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.

3.4) Circuit/Output





3.5) Inference

Here the input is given at port D and output is obtained at port C. The particular example has the numbers 11(Decimal: 3) and 1(Decimal: 1) as input. The output is clearly visible as 11(Decimal: 3) which is the desired output. Hence the multiplication has been carried out successfully.