EE23B022RprtLab3

Deepak Charan S ee23b022

August 2024

1 Introduction

I am Deepak Charan S (Roll No: EE23B022) and this is my report for the second assignment of Microprocessor Lab, which I had done on 27/8/24.

2 Objective:

- To wire the AVR atmgea8 microcontroller along with the given peripherals in a breadboard.
- To program the microcontroller to read the DIP switch values and display it in an LED using assembly programming.
- To 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 LED's connected to another port.
- to program an LED pulse

3 Equipments/Software Required:

1. Atmel AVR (Atmel8L) Chip
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 Microchip Studio 7 and AVR Burn-O-MAT (for burning asm)

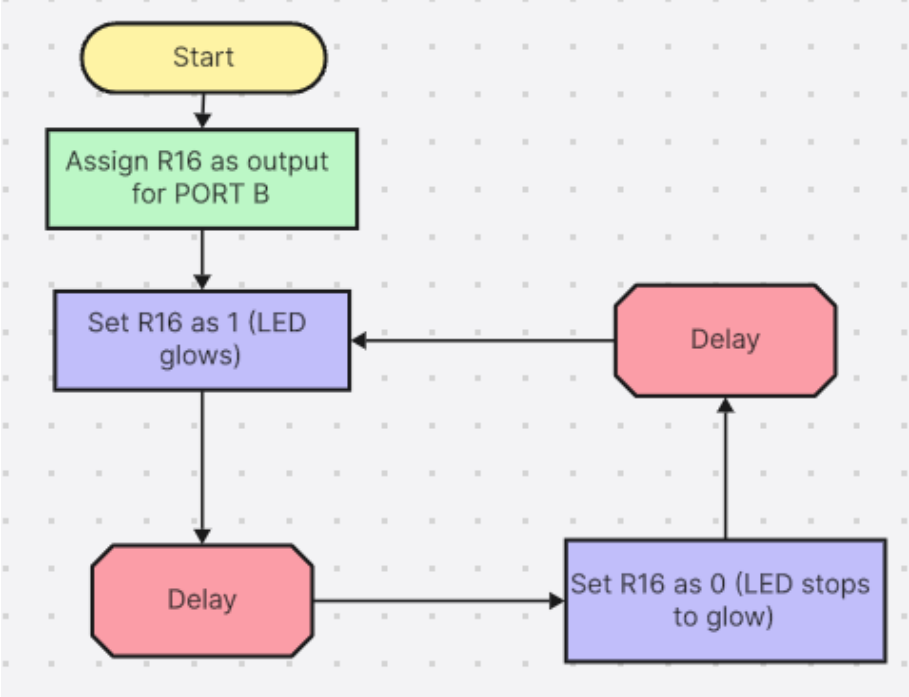
4 General Procedure:

1. Write the AVR assembly code on microchip studio IDE to accomplish all the four objectives
2. Clean existing solutions and build new solution at the start of debugging
3. Debug and test them out line by line on microchip studio and analyse the values in registers, SRAM and carry flags
4. Wire the required components properly on the breadboard
5. Use the AVR Programmer to connect the breadboard and the PC
6. generate the .hex file from Microchip Studio
7. Use the .hex to burn the file into the AVR chip using AVR Burn-O-MAT
8. Test out the hardware to verify the result

5 Problem 1: Blinking an LED

5.1 Implemention:

1. We first initialise the R16 register as an output for PORT D
2. We assign value to 1 so that LED lights up
3. Then we create a nested for loop which does noting but iterate over 5,00,000 times. This is done to create a delay for the LED to toggle to 0
4. set R16 to 0 so that LED stops glowing
5. repeat the delay process
6. loop back to step 2 so that we observe a continuous pulse

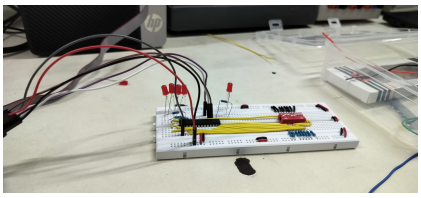


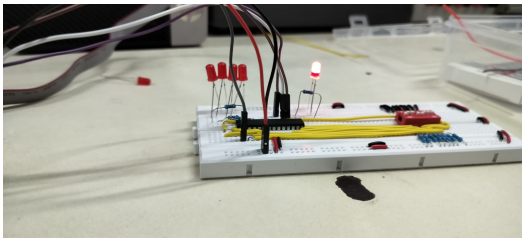
5.2 Code:

```
;pulse.asm

.include "m8def.inc"

LDI R16 , 0xFF ; All bits set as 0 to make port as output
OUT DDRD, R16 ; DDRB = Data Direction Register for PORT D
again:
LDI R16, 0x01
OUT PORTD, R16  ;LED output is HIGH

LDI R17, 0x21 ; Creating a Delay Cycle to give a pulse width of 0.5 sec
LOOP4:LDI R18, 0x32
LOOP5:LDI R19, 0x64
LOOP6:DEC R19
BRNE LOOP6
DEC R18
BRNE LOOP5
```

```
18 DEC R17
19 BRNE LOOP4
20
21 LDI R16, 0x00
22 OUT PORTD, R16   ;LED output is LOW
23
24 LDI R17, 0x21    ;Creating another Delay Cycle
25 LOOP1:LDI R18, 0x32
26 LOOP2:LDI R19, 0x64
27 LOOP3:DEC R19
28 BRNE LOOP3
29 DEC R18
30 BRNE LOOP2
31 DEC R17
32 BRNE LOOP1
33
34 RJMP again       ; Creating an infinite loop to generate
      continuous pulses
```

5.3 Photos:





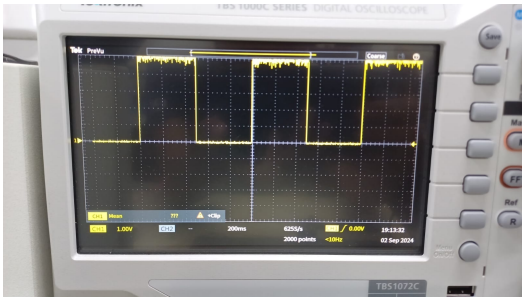


Figure 1: pulse waveform measured using an oscilloscope

6 Problem 2: Controlling an LED using push button

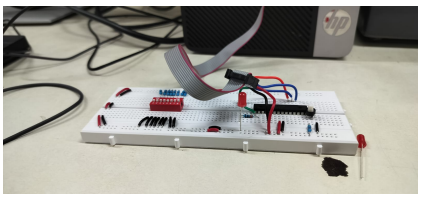
6.1 Implemention:

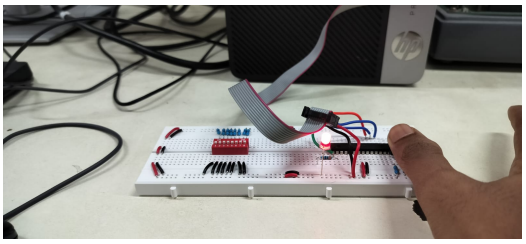
1. We first initialise the R16 register as 0xFF, i.e., make all its bits equal to 1.
2. We write the contents of R16 onto DDRD to make port D an output port.
3. Then we made R16 0x00, i.e, we made all bits of R16 equal to 0.
4. We write the contents of R16 onto DDRB to make port B an input pin.
5. Then we store the input of port B in R16 and write it onto port D. *Note: Since input is low when push button is pressed, we need to invert the R16 value while sending it as output for port D*
6. LED is connected to port D.

6.2 Code:

```
;
; push.asm

.include "m8def.inc"
again: LDI R16 , 0xFF ; All bits set as 1 to make port as output
       OUT DDRD, R16 ; DDRD = Data Direction Register for PORT D
       LDI R16 , 0x00 ; All bits set as 0 to make port as input
       OUT DDRB, R16 ; DDRB = Data Direction Register for PORT B
       IN R16 , PINB ; Store the input of PORT B to register R16
       COM R16 ; reason mentioned in implementation section
       OUT PORTD, R16 ; Set LED connected to PDx as input of PBx
       RJMP again
```

6.3 Photos:

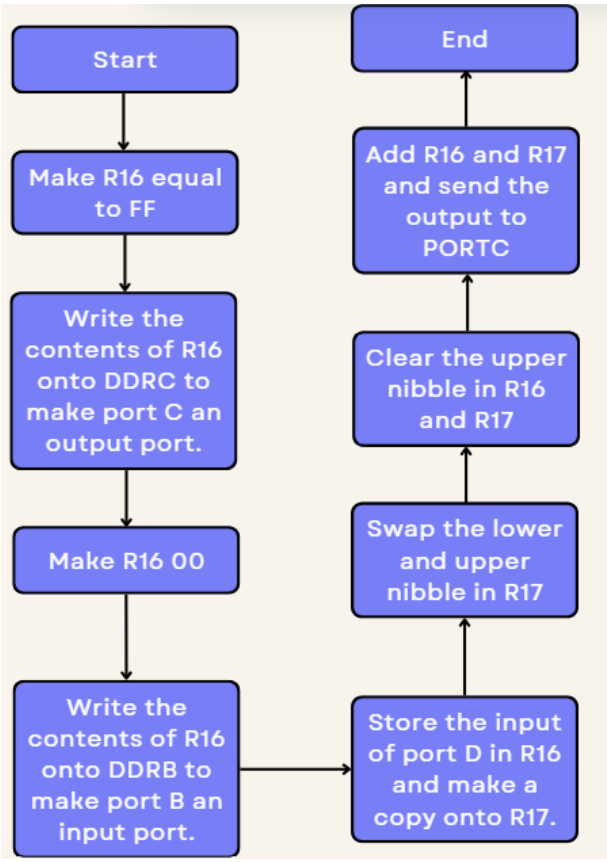




7 Problem 3: Displaying 4 bit addition

7.1 Implemention:

1. We first initialise the R16 register as 0xFF, i.e., make all its bits equal to 1.
2. We write the contents of R16 onto DDRC to make port C an output port.
3. Then we made R16 0x00, i.e, we made all bits of R16 equal to 0.
4. We write the contents of R16 onto DDRD to make port D an input pin.
5. DIP switches are connected to port D.
6. Then we store the input of port D in R16 and make a copy onto R17.
7. Swap the lower and upper nibble in R17
8. Clear the upper nibble in R16 and R17 to get first 4 bits and last 4 bits from DIP switches in two separate registers
9. Add R16 and R17 and send the output to PORTC
10. 5 LEDs are connected to port C.

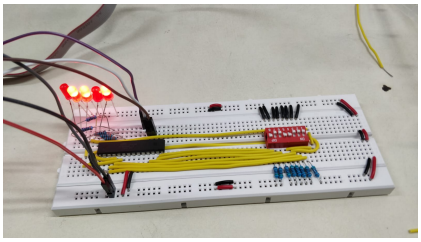


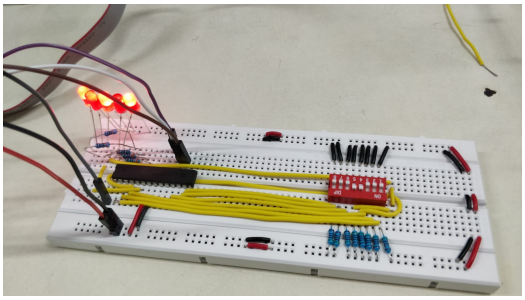
7.2 Code:

```
; 4bit_add.asm

.include "m8def.inc "
again: LDI R16, 0xFF ; All bits set as 1 to make port as
    output
        LDI R17, 0x00 ; Clearing register for later use
        OUT DDRC, R16 ; DDRC = Data Direction Register
           for PORT C
        LDI R16, 0x00 ; All bits set as 0 to make port
           as input
        OUT DDRD, R16 ; DDRD = Data Direction Register
           for PORT D
        IN R16, PIND ; Store the input of PORT D to
           register R0
        MOV R17, R16 ; Copy input to R1
        ANDI R16, 0x0F; Make R0 only first 4 bits
```

```
12          SWAP R17 ;
13          ANDI R17, 0x0F; Make R1 only first 4 bits
14          ADD R16, R17 ; Add and store result in R0
15          OUT PORTC, R16 ; Set LED connected to PCx as
               input of PDx
16          RJMP again
```

7.3 Photos:





8 Interpretations of result

Controlling AVR Peripherals through Assembly Programming was demonstrated through a breadboard circuit.

Note: The code for blinking LED was done by me.

9 References:

Handouts and Instruction Manual of AVR provided in moodle. Slight reference from Mazidi's "The AVR Microcontroller and embedded systems" was also used.