MICROPROCESSOR LAB EXPERIMENT 5

GROUP - 18
Deenabandhan N ee23b021
Sai Harshith Gajendra ee23b069
Krutarth Patel ee23b137

27 September 2024

Introduction :

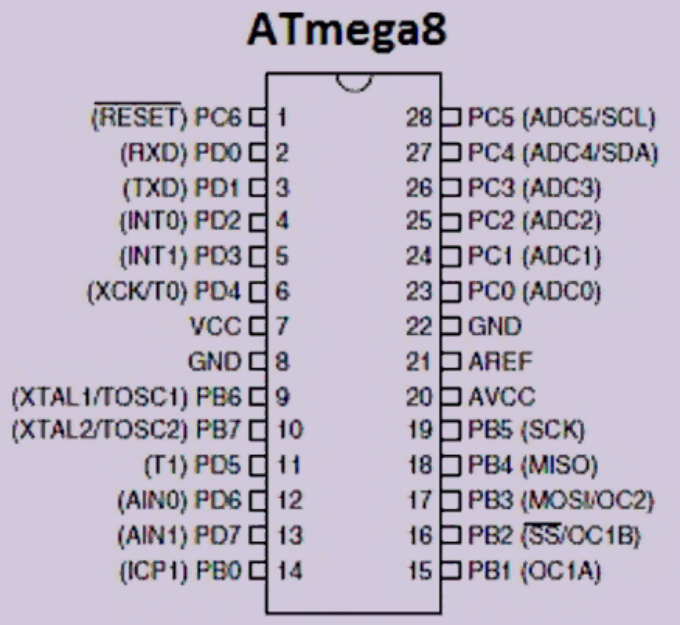
- In this experiment , we are going to learn about interrupts and subroutines in C and Assembly programs using the **ATmega8**.
- This experiment involves ,
  - Introduction to Interrupt handling.
  - Writing a program in C to transfer control from a white LED(turned on) to a blinking LED on a button press
  - Writing a program in Assembly to blink an LED upon receiving an interrupt in the form of a button press
  - Changing the code in the above task to transfer control from a white LED(turned on) to a blinking LED on a button press
- In this report , we have included the code of the tasks and our experience with C and Assembly.

ATmega-8 and Microchip studio :

- Atmega-8 is an 8-bit RISC single-chip microcontroller developed by Atmel.
- The number 8 in its name represents that it can operate 8 bits at a time while processing the information i.e in a way it represents the capacity of the microcontroller.
- Some features of AVR microcontroller are
  - I/O ports.
  - Internal instructions flash memory
  - SRAM upto 16KB
  - Timers
- The AtMega8 microcontroller has a total of 32 8-bit registers and 23 I/O pins.

Atmega8 microcontroller pin diagram :

- The pin diagram of Atmega8 microcontroller is ,



- It has 3 ports: PortB, PortC and PortD.
- Each port acts as a bidirectional buffer that could carry both input and output values with specific address.
- The registers that are associated with these ports are
  - **DDRX** - Register to mention whether the particular pin is input/output. Eg : DDRD=0x0F means , first 8 pins are output pins and the rest are input pins.
  - **PORTX** - Register to mention the output to be given through the pin. Eg : PORTC=0xF0 means that the first 8 pins of Port C are set to logic low and the rest of them are set to logic high.
  - **PINX** - Register that is used to store the value thta is given as input in the pins. Eg : a=PINB means that whatever input that is given at port B is given to the variable a.
- In addition to these ports it also supports interrupt operations which is an important instruction in any microcontroller.

Libraries used in the C code

```
#include <avr/io.h>
```

- The above library is used to include standard avr commands like **DDRD , PORTC , PINB**

```
#include <util/delay.h>
```

- The above library is used to include time delays using the function ,

```
delay_ms(100) //includes 100ms delay
```

Interrupts in C

Introduction :

- This task involves writing a C program to transfer control from a white LED(turned on) to a blinking LED on a button press
- pressing a button will send an interrupt signal to the program which will then run the subroutine we have written to turn off the white LED and blink the other LED at a constant frequency.

Code

```
1
2  #define F_CPU 8000000UL
3  #include <avr/io.h>
4  #include<util/delay.h>
5  #include <avr/interrupt.h>
6  int main(void)
7  {
8    DDRB=0x03; // LED connected as output
9    DDRD=0x00; // input
10   GICR=0x40; // setting INT0 interrupt
11   SREG=0x80; // global interrupt enable
12   while(1)
13   {
14     PORTB=0x01; // turning on LED
15   }
16 }
17 ISR(INT0_vect)
18 {
19   cli(); // disabling interrupts
20   PORTB=0x02; // switching LED
21   _delay_ms(100); //blinking logic
22   PORTB=0x00;
23   _delay_ms(100);
24   sei(); //enabling interrupts
25 }
```

With PORTC

Code for a single LED :

```c
#include <avr/io.h>

int main(void)
{
    DDRC=0x1F; // Making all the pins of port C to be output
    while (1)
    {
        PORTC=0x01; //Making the PortC , Pin C0 to be active high
    }
}
```

Code for a single LED with delay :

```c
#include <avr/io.h>
#include <util/delay.h> //To include the delay function
int main(void)
{
    DDRC=0xFF; // Making all the pins of port D to be output
    while (1)
    {
        PORTC=0x01; //Making the PortC , Pin C0 to be active high i.e on
        _delay_ms(1000); // Causing 1 second gap
        PORTC=0x00; //Making the PortD , Pin C0 to be active low i.e off
        _delay_ms(1000); // Causing 1 second gap
  }
}
```

Code for two LEDs to blink alternately :

Code :

```c
#include <avr/io.h>
#include <util/delay.h> //To include the delay function
int main(void)
{
    DDRC=0xFF; // Making all the pins of port C to be output
    while (1)
    {
        PORTC=0x01; //Making the PortC , Pin C0 to be active high and C1 to be active low i.e on and off
        _delay_ms(1000); // Causing 1 second gap
        PORTC=0x02; //Making the PortC , Pin C0 to be active low and C1 to be active high i.e off and on
        _delay_ms(1000); // Causing 1 second gap
  }
}
```

Explanation :

- In this code , two pins of **PORTC** i.e pin C0 and pin C1 are connected to LEDs.
- So the state of the LEDs will be 01 an 10.
- To make this happen the PORTC variable will have 0x01 and 0x02 states at the interval of 1 second.

Code for two LEDs to repeat the given sequence :

Code :

```c
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRC=0xFF; // Making all the pins of port C to be output
    unsigned char count=0; //counter to perform the task
    while (1)
    {
        PORTC=(count%4); //Making the PortC respective pin to be high based on the count modulo 4
        _delay_ms(1000); // Causing 1 second gap
        count=count+1; //Updating the counter
    }
}
```

Explanation :

- In this code , two pins of **PORTC** i.e pin C0 and pin C1 are connected to LEDs.
- So the state of the LEDs should be either 00,01,10 and 11.
- In hexadecimal , the states can be represented as 0x00,0x01,0x02 and 0x03.
- For this , we use a counter variable to achieve these states.
- So after each loop we update this counter variable and take modulo 4 of this variable to assign it to PORTC.

Johnson Counter

Introduction :

- This task involves adding generating a Johnson counter

Code

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

int main(void)
{
    DDRC = 0x07;  // Set lower 3 bits of PORTC as output (for 3-bit Johnson counter)
    uint8_t counter = 0b000;  // Initialize the 3-bit counter to 000

    while (1)
    {
        PORTC = (PORTC & 0xF8) | (counter & 0x07);  // Output lower 3 bits to PORTC, keep upper 5 bits unchanged

        // Johnson counter mechanism: shift right and invert the new bit
        counter = (counter >> 1) | ((~(counter & 0x01)) << 2);

        _delay_ms(500);  // 500ms delay for visualizing the output
    }
}
```

Explanation :

- Counter is initially set to 0.
- The bits are shifted right and the new bit is inverted.

4-bit Addition

Introduction :

- This task involves adding two 4-bit numbers and displaying the result using LEDs connected to PORT-C

Code

```c
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>
int main(void)
{
  DDRC = 0x1F; //output port
  unsigned char a = 15;
  unsigned char b = 200;

  while(1)
  {
    PORTC = a + b;
    _delay_ms(10);
  }
}
```

Process

- Adding the two numbers.
- Displaying the result using LEDs.

Bonus Question

Introduction :

- This task involves taking two 4-bit numbers from PORT-D and displaying their sum using LEDs connected to PORT-C.

Code

```c
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>
int main(void)
{
  DDRC = 0x1F; //output port
  DDRD = 0x00; //input port

  while(1)
  {
    unsigned char data = PIND; //receiving data

    // we get the 4-bit numbers as one 8-bit number
    // eg: number1 ->0101 1110<- number2
    PORTC = (data>>4)+ (data&15);
    _delay_ms(10);
  }
}
```

Process

- Getting two 4-bit numbers from PORT-C.
- Adding the two numbers.
- Displaying the result using LEDs.

Result