EE2016 Microprocessor Lab & Theory -July - November 2022

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

1 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 LED's connected to another port.

2 Equipments, Hardware Required

To perform this experiment, the following components are required.
1. Atmel AVR (Atmel8L) Chip - 1
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-O-MAT (for burning _ash)

3 Concepts required:

3.1 Configuration of Ports

The architecture of AVR microcontrollers is register-based: information in the microcontroller such as the program memory, state of input pins and state of output pins is stored in 1 registers. There are a total of 32 8-bit registers. Atmega8 has 23 I/O pins. These pins are grouped under what are known as ports. A port can be visualized as a gate with a specific address between the CPU and the external world. The CPU uses these ports to read input from and write output into them. The Atmega8 microcontroller has 3 ports: PortB, PortC, and PortD. Each of these ports is associated with 3 registers - DDRx, PORTx and PINx which set, respectively, the direction, input and output functionality of the ports. Each bit in these registers configures a pin of a port. Bit0 of a register is mapped to Pin0 of a particular port, Bit1 to Pin1, Bit2 to Pin2 and so on. Each register is explained in detail below.

3.1.1 Register DDRx

DDR stands for Data Direction Register and x indicates a port alphabet. As the name suggests, this register is used to set the direction of port pins to either input or output. For input, we set to 0 while for output, we set to 1. For instance, let us consider PortB. To set this port as input or output, we need to initialize DDRB. Each bit in DDRB corresponds to the respective pin in PortB. Suppose we write DDRB = 0xFF, then all bits in PortB are configured to Output. Similarly, DDRB=0x00 configures the port to be Input.

3.1.2 Register PORTx

PORTx sets a value to the corresponding pin. DDRx can set a bit to be either input or output while PORTx changes its functionality based on it. Consider the case where pins are configured as output pins. Suppose all pins are set to be output. This corresponds to DDRx = 0xFF. Now, writing 1 to a bit in PORTx will make the output high (normally +5 volts) for that particular pin. Writing 0, on the other hand, makes the output low (0 volts) in that particular pin. Suppose a particular bit in DDRx is set as Input (0), then there are two possibilities. If corresponding PORTx bit is set to 1, then an internal pull-up resistor is enabled. This implies that even if we do not connect this pin to anything, it will still read as 1. However, this can be made to read as 0 by externally connecting it to ground and pulling it down. An interesting situation happens if PORTx is set to 0 and pins are set as Input. In this case, the pin enters a tri-state mode and the internal pull-up resistor is disabled. If the pin is not connected, it picks up noise from the surroundings and the (read) result from this pin is unpredictable. It is always a good practice to enable the internal pull-up resistor while reading.

3.1.3 Register PINx

The functionality of a PINx register is straightforward: it reads data from the port pin. If DDRx is set to Input (0) for a particular bit, then we can read data from PINx. However if DDRx is set to Output (1), then reading PINx register gives the same data which has been output on that particular pin.

3.2 Assembly Programming

AVR microarchitecture supports special instructions to interact with peripheral devices. Two such instructions are IN and OUT instruction.

3.2.1 OUT

This instruction stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers, etc.). For example, "OUT A, Rx" writes the value stored in Rx to I/O register named A.

3.2.2 IN

This instruction loads data from the I/O Space (Ports, Timers, Configuration Registers, etc.) into register Rd in the Register File. For example, "IN Rd, A" loads the value stored in I/O register named A to register Rd.

3.2.3 RJMP

This instruction is used to jump the control of the program to a particular instruction. For example, the following snippet moves the program counter to the instruction LDI after execution of MOV.

```
again: LDI R16, 0x23
       MOV R0, R1
       RJMP again
```

3.2.4 Other Instructions

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

Refer Reference 3 for exhaustive list of all instructions

3.3 Sample Program

The following program can be used to read from 8 input pins in PORTB and write to the corresponding numbered pins in PORTD.

```
#include " m8def.inc "
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 , 0 x00 ;         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
OUT PORTD, R16 ;          Set LED connected to PDx as input of PBx
```

4 Procedure

4.1 Programming Environment

1. Open Atmel Studio 6.2 software installed in windows.
2. Create a new project with template as Assembler and select AVR Assembler Project

3. Set device family as Atmega AVR, 8-bit and select the device ATmega8. You should find a new assembler project being created.
4. Write the assembly program for reading from DIP switches and writing to LEDs
5. Under the Build menu, select Build Solution to compile and build your project. If the program is correct, you will find a .HEX file being created in Debug folder.

4.2 Connections

1. Wire the Atmega8 processor in the breadboard with AVR Programmer as shown in Figure 1.

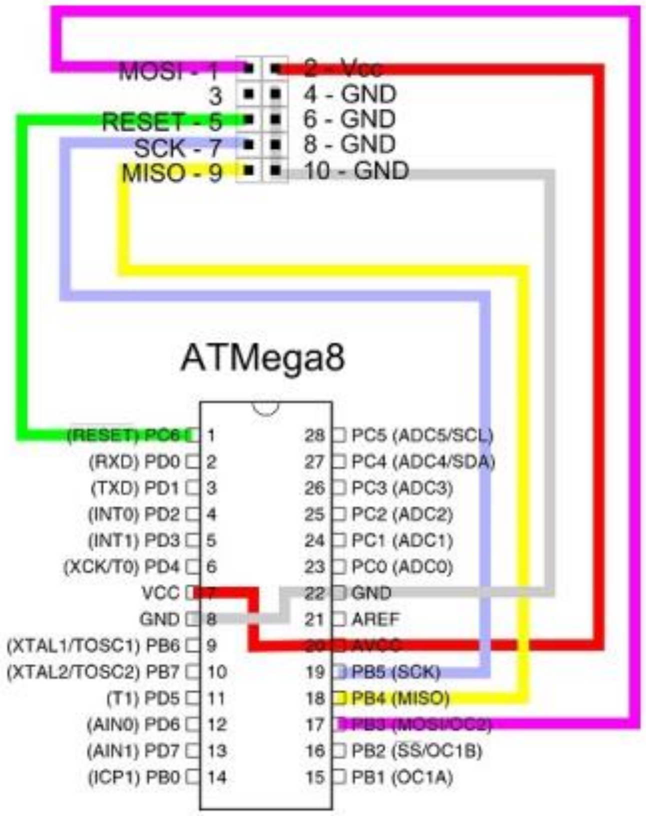

Figure 1: Microcontroller to In-System Programmer Connections

2. Connect any input port, say PB0 to one end of DIP switch and other end to ground. Connect a 10 kW pull-up resistor to PB0 as shown in Figure 2.

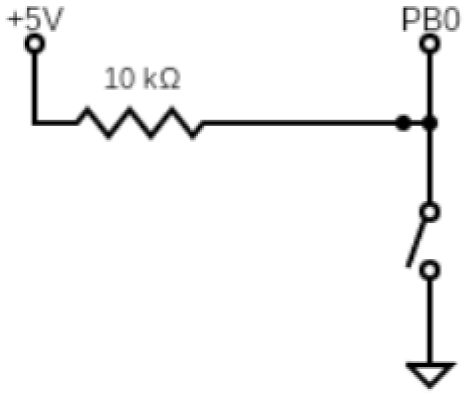

Figure 2: Connection to input port 5

3. Connect an LED between output port, say PD0 and ground preferably through a 300Ω resistor.

4.3 Burning the program

1. Connect the USB cable to the CPU.
2. Open AVR Burn-O-MAT software and select AVR type as ATmega8
3. Under the flash section, select the previously generated .HEX file. Click on write to write the program to the microcontroller.
4. Demonstrate the working model to the TAs and verify the results for the different programs.

5 Problems:

1. Follow the above procedure to control an LED using a DIP switch
2. 4 bit addition of two unsigned nibbles from the 8 bit dip input switch and display the result obtained in LEDs.
3. 4 bit multiplication of two unsigned nibbles and display the result in LEDs.

6 References

1. ATmega8(L) - Summary Datasheet
2. ATmega8(L) - Complete Datasheet
3. AVR instruction set manual
4. Muhammad Ali Mazidi, Sarmad Naimi and Sepehr Naimi, Chapter4, AVR I/O Port Programming, the avr microcontroller and embedded system using assembly and c