MICROPROCESSOR LAB EXPERIMENT 6

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

18 September 2024

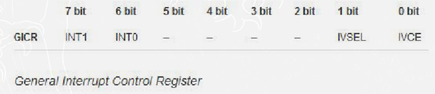
Microprocessor Lab Experiment 6

Introduction :

- This experiment tries to convey about the interrupt option available in microcontroller that helps us analyse the systems in microcontroller.
- The General Interrupt Control Register (GICR) is an 8-bit register.
- The 6th bit (7th from the right end) is set to 1 to enable INT0
- The remaining bits are set to 0.
- IVSEL and IVCE correspond to Interrupt Vector Select and Interrupt Vector Change Enable and these are set to 0 to allow interrupts.
- Setting status register SREG to 0x80 corresponds to global interrupt enable.
- Within the ISR, we note that CLI clears the global interrupt flag (I) in the status register (SREG) and thereby disables the interrupts. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with CLI.
- While coming out of the ISR, we have an SEI which sets the global interrupt flag (I) to 1.

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.



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
GICR	INT1	INT0	–	–	–	–	IVSEL	IVCE

General Interrupt Control Register

Figure 1: Structure of GICR

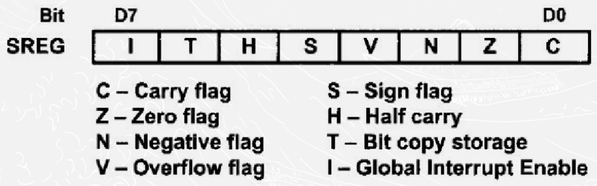
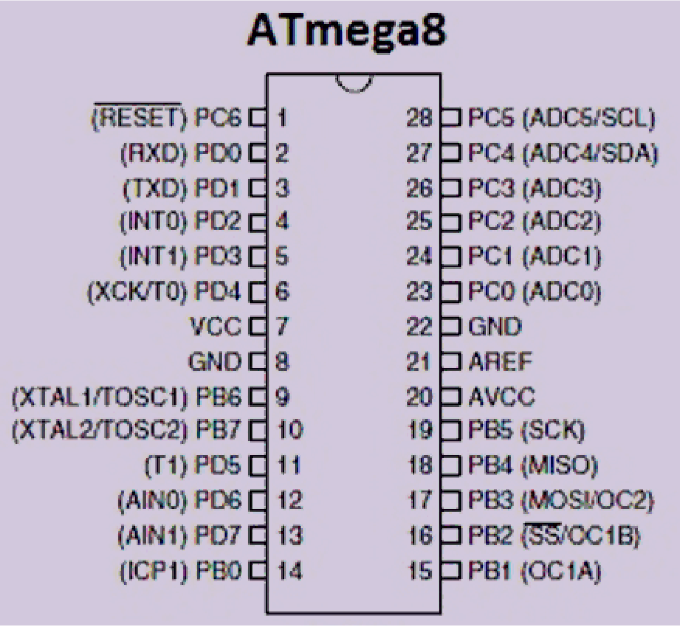


Figure 2: Structure of status register

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
```

Microprocessor Lab Experiment 6

Instructions used :

Instruction	Usage
LDI Rx,Rd	Load the value **Rd** in the register **Rx**
LD Rx,Rm	Load the value stored in the memory address **Rm** in the register **Rx**
LPM Rx,Rm	Load the value stored in the program memory address **Rm** in the register **Rx**
ST Rm,Rx	Store the address of the register **Rx** in the pointer **Rm**
MOV Rm1,Rm2	Copy the value stored in the register **Rm1** to the register **Rm2**
SUB Rm1,Rm2	Subtract the value stored in **Rm1** from the value stored in **Rm2** register and store the result in **Rm1**
CP Rm1,Rm2	Compare the values of the registers **Rm1** and **Rm2** and raise the following flags in the status registers - Sign flag - Negative flag - Carry flag if the first value is smaller than the second value.
DEC Rm	Decrease the value in the register **Rm** by an unit.

Microprocessor Lab Experiment

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

```

#define F_CPU 8000000UL
#include <avr/io.h>
#include<util/delay.h>
#include <avr/interrupt.h>
int main(void)
{
  DDRB=0x03; // LED connected as output
  DDRD=0x00; // input
  GICR=0x40; // setting INT0 interrupt
  SREG=0x80; // global interrupt enable
  while(1)
  {
    PORTB=0x01; // turning on LED
  }
}
ISR(INT0_vect)
{
  cli(); // disabling interrupts
  PORTB=0x02; // switching LED
  _delay_ms(100); //blinking logic
  PORTB=0x00;
  _delay_ms(100);
  sei(); //enabling interrupts
}
```

Explanation

- We first set the required pins in PORTB and PORTD to output. setting PORTB to 0x03 sets PINB0 and PINB1 to output
- GICR or General Interrupt Control Register allows us to enable external interrupts
- SREG or Status Register allows us to enable global interrupts

Microprocessor Lab

Task-2

Introduction :

- In this task , we will write an assembly code for implementing the led blinking 10 times.
- The task aims to make a clear understanding of using interrupts in Atmega8.
- We can decompose the code to multiple parts.
- In this report , let us divide the code into two parts i.e main loop , interrupt part.

Main loop:

Code

```
        .org 0
                rjmp reset ; on reset, program starts here
        .org 0x002 ; Interrupt vector address for INT1.
                rjmp int1_ISR ;
        reset:
                ldi R16,0x70 ; setup the stack pointer to point to address 0x0070
                out spl,R16
                ldi R16,0x00
                out sph,R16

                ldi R16,0xFF ; make PB1 output
                out DDRB,R16 ; R16 values are given to PortB's DDRB


                ldi R16,0x00; make PORTD input
                out DDRD,R16

                ldi R16,0x08 ; use pull-up resistor for PD3
                out PORTD,R16

                in R16,GICR
                ori R16,0x80 ; enable INT1 interrupt
                out GICR,R16

                ldi R16,0x00; Turn off LED on PB1
                out DDRB,R16
                mov R20,R16;
                SEI ; enable interrupts

        ;indefiniteloop: RJMP indefiniteloop

```

Microprocessor Lab Experiment 6

Working part :

- In the main loop , we are doing a bunch of operation step by step to achieve interrupt operations.
- Let's analyse each operation individually.

Specifying the vector address :

```
        .org 0
                rjmp reset ; on reset, program starts here
        .org 0x002 ; Interrupt vector address for INT1.
                rjmp int1_ISR ;
```

- The above code specifies the code where to go when interrupt signal is passed or it comes to reset.

- The first .org command memory location of reset is 0x00 which is jump over the vector table.
- The next

Creating a stack pointer :

```
        reset:
                ldi R16,0x70 ; setup the stack pointer to point to address 0x0070
                out spl,R16
                ldi R16,0x00
                out sph,R16
```

- This part of the code sets up the stack pointer which may be used to store values by pushing in and gt back by pulling the last data out.
- We are assigning the memory location 0x0070 to the stack pointer memory specified by **spl** and **sph**.

Creating outputs and inputs :

```
        ldi R16,0xFF ; make PB1 output
        out DDRB,R16 ; R16 values are given to PortB's DDRB

        ldi R16,0x00; make PORTD input
        out DDRD,R16

        ldi R16,0x08 ; use pull-up resistor for PD3
        out PORTD,R16
```

- This part of the code gives the input and output control values.
- We are assigning the **DDRB** to 0xFF as it is an output.
- We are assigning the **DDRD** to 0x00 as it is an input.
- We are also assigning the 3 pin of Port D to have an active low value as **PD3** will have the interrupt pin.

Enabling interrupts and turning off LED :

```
        in R16,GICR
        ori R16,0x80 ; enable INT1 interrupt
        out GICR,R16

        ldi R16,0x00; Turn off LED on PB1
        out DDRB,R16

        SEI ; enable interrupts
```

- This part of the code enables the interrupt **INT_1** by assigning the GICR to have the value 0x80.
- We will Turn off the LED on PB1 by assigning it 0x00.
- The command **SEI** is used to enable the interrupts.

- The indefinite loop is used to execute the code infinite times.

Interrupt part:

Code :

```
    int1_ISR: ; INT1 interrupt handler or ISR
        CLI; clear interrupts
        in R16,SREG; save status register SREG
        push R16

        ldi R16, 0x0A; blink led 10 times
```

Microprocessor Lab Experiment 6

```
    mov R0,R10
    back5:
            ldi R16,0x02 ; Turning on LED on PB1
            out PORTB,R16
            delay1:
                            LDI R16,0xFF ; delay
                            back2:
                                    LDI R17,0xFF
                                    back1: DEC R17
                                    BRNE back1
                                    DEC R16
                            BRNE back2
                    ldi R16,0x00; Turn off LED on PB1
                    out PORTB,R16
            delay2:
                            LDI R16,0xFF; delay
                            back3:
                LDI R17,0xFF
                              back4: DEC R17
                              BRNE back4
                              DEC R16
                            BRNE back3
            DEC R0
    BRNE back5; ; check if LED has blinked 10 times

    pop R16 ; retrieve status register
    out SREG,R16
    reti; go back to main program
```

Working :

Saving status register and starting the loop :

```
int1_ISR: ; INT1 interrupt handler or ISR
    CLI; clear interrupts
    in R16,SREG; save status register SREG
    push R16

    ldi R16, 0x0A; blink led 10 times
    mov R0,R10
```

- This part of the code uses the stack pointer to save the current status register in the stack pointer.
- We have the counter variable as **R0** here.

Looping statement and turning on with a delay :

```
    ldi R16,0x02 ; Turning on LED on PB1
        out PORTB,R16
        delay1:
                        LDI R16,0xFF ; delay
                        back2:
                                LDI R17,0xFF
                                back1: DEC R17
                                BRNE back1
                                DEC R16
                        BRNE back2
```

- This part of the code is used to turn on the LED by assigning PORTB 0x02.
- To add the delay to the blinking we have delay 1 where basically 255*255 operations are performed by having counters as **R16** and **R17**.
- The delay is caused by the labels **back1** and **back2**

Microprocessor Lab Experiment 6

Looping statement and turning off with a delay :

```
    ldi R16,0x00; Turn off LED on PB1
        out PORTB,R16
    delay2:
                        LDI R16,0xFF; delay
                        back3:
            LDI R17,0xFF
                          back4: DEC R17
                          BRNE back4
                          DEC R16
                        BRNE back3
        DEC R0
BRNE back5; ; check if LED has blinked 10 times
```

- This part of the code is used to turn off the LED by assigning PORTB 0x00.
- To add the delay to the blinking we have delay 1 where basically 255*255 operatios are performed by having counters as **R16** and **R17**.
- The delay is caused by the labels **back3** and **back4**.
- Besides these the counter **R0** counts the number of blinking by the label **back5**.

Retrieving status register and returning to main loop :

```
pop R16 ; retrieve status register
out SREG,R16
reti; go back to main program
```

- This part of the code uses the stack pointer to retrieve the status register stored in the stack pointer.
- The command **reti** is used to jump back to main program after enabling the interrupt.

Microprocessor Lab Experiment 6

Task-3

Introduction :

- In this task , we will write an assembly code for implementing the task-1.
- The task aims to make a clear understanding of using interrupts in Atmega8.
- We can decompose the code to multiple parts.
- In this report , let us divide the code into two parts i.e main loop , interrupt part.

Main loop:

Code

```
        .org 0
                rjmp reset ; on reset, program starts here
        .org 0x002 ; Interrupt vector address for INT1.
                rjmp int1_ISR ;
        reset:
                ldi R16,0x70 ; setup the stack pointer to point to address 0x0070
                out spl,R16
                ldi R16,0x00
                out sph,R16

                ldi R16,0xFF ; make PB1 output
                out DDRB,R16 ; R16 values are given to PortB's DDRB


                ldi R16,0x00; make PORTD input
                out DDRD,R16

                ldi R16,0x08 ; use pull-up resistor for PD3
                out PORTD,R16

                in R16,GICR
                ori R16,0x80 ; enable INT1 interrupt
                out GICR,R16

                ldi R16,0x01; Turn off LED on PB1 and turn on LED on PB0
                out DDRB,R16

                SEI ; enable interrupts

        ;indefiniteloop: RJMP indefiniteloop

```

Changes from task-2:

- Here we assign **PORTB** to have two outputs PB1 and PB0.

```
        ldi R16,0x01; Turn off LED on PB1 and turn on LED on PB0
        out DDRB,R16
```

Interrupt part:

Code :

```
    int1_ISR: ; INT1 interrupt handler or ISR
        CLI; clear interrupts
        in R16,SREG; save status register SREG
```

Microprocessor Lab Experiment 6

```
        push R16

        ldi R16, 0x0A; blink led 10 times
        mov R0,R10
        back5:
                ldi R16,0x02 ; Turning on LED on PB1 and turning off LED on PB0
                out PORTB,R16
                delay1:
                                LDI R16,0xFF ; delay
                                back2:
                                        LDI R17,0xFF
                                        back1: DEC R17
                                        BRNE back1
                                        DEC R16
                                BRNE back2
                        ldi R16,0x00; Turn off LED on PB1
                        out PORTB,R16
                delay2:
                                LDI R16,0xFF; delay
                                back3:
                    LDI R17,0xFF
                                  back4: DEC R17
                                  BRNE back4
                                  DEC R16
                                BRNE back3
                DEC R0
        BRNE back5; ; check if LED has blinked 10 times

        pop R16 ; retrieve status register
        out SREG,R16
        reti; go back to main program
```

Changes :

- There is no change in interrupt loop.

Microprocessor Lab Experiment 6