Lecture 9 Interrupt Processing

Objectives:

1. Understand the different type of data transfer modes between microprocessor and I/Os
2. Understand the general interrupt operation sequence and procedure in microprocessors
3. Understand the ATmega328 interrupt features.
4. Learn to configure and write interrupt programs for ATmega328P in C
5. Understand how to employ the interrupt system as an components for a microcontroller based system.

Contents (Chapter 9 in the textbook)

1. Input/Output mode

The data transfer between the microprocessor and input/output unit can be divided into two categories: the microprocessor initiated and I/O device initiated data transfer.

In microprocessor initiated mode, the microprocessor will initiate the communication to read/send data from/to the input/output devices. I/O device can always be ready for communication, such as the LEDs connected to I/O port. This mode is called unconditional MPU microprocessor initiated data transfer. Some devices may not always be ready. For example, the microprocessor has to keep checking the ADC till an ADC completes the conversion. This is called polling. Only when the I/O device is ready, it may process the data transfer.

I/O device initiated data transfers include interrupt transfer and Direct Memory Access (DMA). In DMA, data transfer occurs between the I/O devices and RAM memory directly, without involving the microprocessor. In interrupt transfers, an I/O device sends a signal to microprocessor's interrupt inputs and microprocessor will interrupt the main program and jump to the interrupt service routine.

In our previous lectures, the data transfers between I/O devices and microprocessors all belong to the microprocessor initiated mode and the program executed in exactly predefined orders/sequences. In this lecture, we will introduce interrupt mode.

2. Interrupt Operations

The interrupt system on a microcontroller allows it to respond to higher priority events such as loss of power or events whose occurrence is unknown in advance, such as keypad or pushbutton inputs. When an interrupt event occurs, the microcontroller will complete the instruction it is currently executing and then jump to the program that controls interrupt event specific tasks. These tasks, which resolve the interrupt event, are organized into a function called an interrupt service routine (ISR). Once the ISR is complete, the microcontroller will resume processing where it left off before the interrupt event occurred.

Before the microprocessor jumps to the ISR, the program environment needs to be saved so that the microprocessor can go back to the original point and resume the processing in the main program. The program environment, or context, in generally, consists of the contents of register sections. Typically, the contents in related registers will be stored by pushing into a stack. A stack is a block of memories that has the feature of First In Last Out (FILO). The registers will be restored by popping data out of the stack after the ISR has been executed.

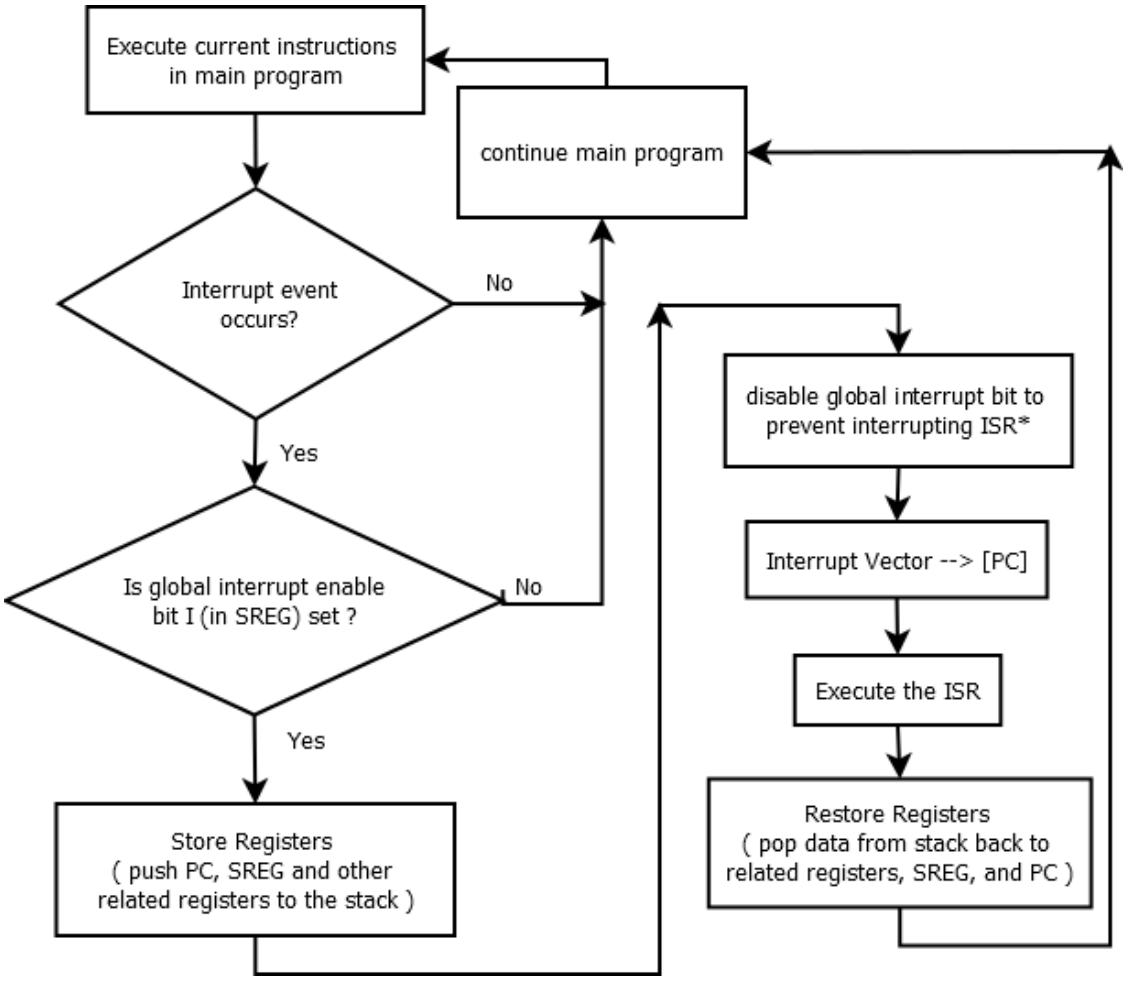

Figure 1 Interrupt handling procedure (*the global interrupt bit may not be cleared if nested ISR is allowed)

3. Interrupts in ATmega328p

There are a total of 26 interrupt sources in ATmega328P, from external interrupts, I/O pin status change, Timer/Counters, Serial Communications, to ADC and EEPROM access. Table 1 gives all interrupt sources in ATmega328P. The starting address of an ISR, called an Interrupt Address Vector, is stored in the specific location that is indicated by the table. When an interrupt event occurs, the corresponding Interrupt Address Vector will be loaded to PC automatically to start executing the ISR.

The procedure of handling an interrupt request is shown in Figure 1 above.

Table 1 Reset and Interrupt Vector in ATmega328P

VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

External Interrupts:

The External Interrupts are triggered by the INT0 and INT1 pins or any of the PCINT23..0 pins. Note that, the interrupts will trigger even if the INT0 and INT1 or PCINT23..0 pins are configured as outputs if enabled.

A. INT0 and INT1 (pin 2 and 3 in Arduino UNO board)
External interrupt sense inputs INT0 and INT1 can be configured as edge triggered or level trigger. Related registers are listed below.

External Interrupt Control Register (EICRA)

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The sensor control bits for ISC11..0 is for INT1 and ISC01..0 are for INT0 and both follow the functions shown in Table given below.

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

External Interrupt Mask Register (EIMSK)

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

When the INT0/1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled.

External Interrupt Flag Register (EIFR)

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

When an edge or logic change on the INT0/1 pin triggers an interrupt request, INTF0/1 becomes set(one). If the I-bit in SREG and the INT0/1 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the

interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT0/1 is configured as a level interrupt.

B. PCINT23..0 (all I/O pins in port B, C, D)
Another sets of external interrupts can be activated by pin status change in any I/) pins in the ports. Note that the interrupt event occurs whenever the pin status changes (can be falling or rising edge).

Pin Change Interrupt Control Register (PCICR)

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

When the PCIE2/PCIE1/PCIE0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 2/1/0 is enabled.

PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

When a logic change on any PCINT23..16 /PCINT14..8 / PCINT7..0 pin triggers an interrupt request, PCIF2/ PCIF1/ PCIF0 becomes set(one). If the I-bit in SREG and the PCIE2/ PCIE1/ PCIE0 bit in PCICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

PCMSK2/1/0 – Pin Change Mask Register 2/1/0

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Each bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If the bit is set and the corresponding PCIE2/1/0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. Otherwise pin change interrupt is disabled.

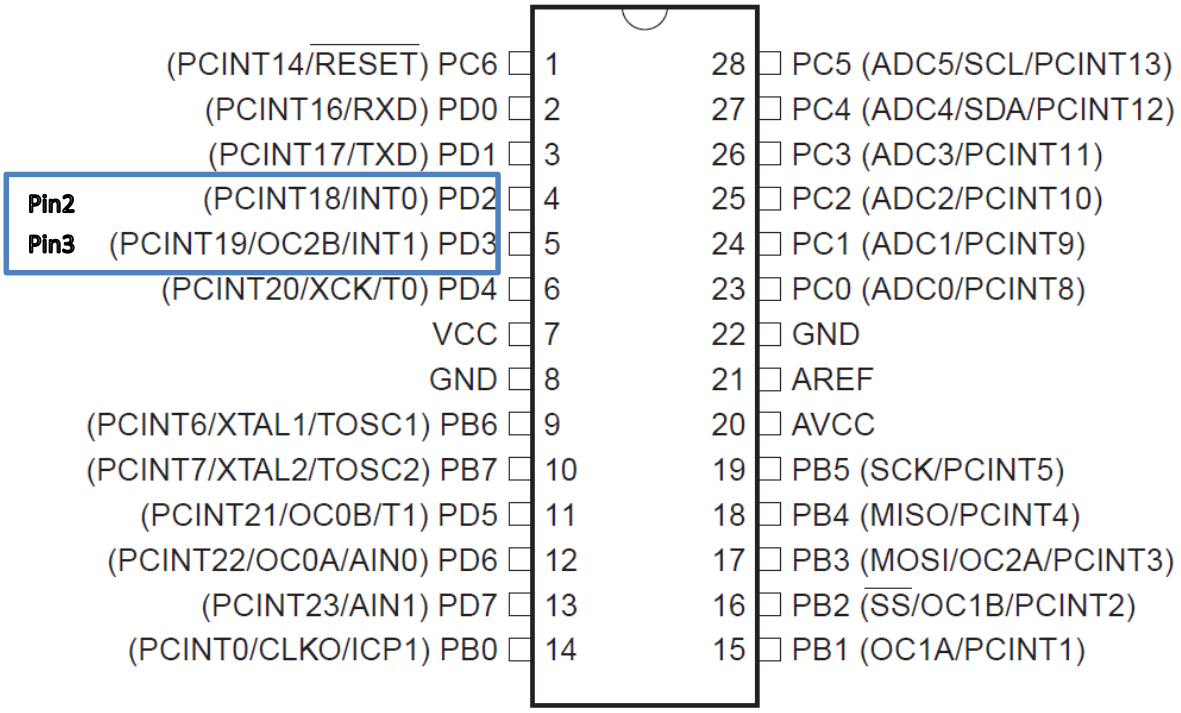

Figure 2 Pin layout in ATmega328P

4. Program an interrupt event in ATmega328P in C

Most C compilers provide special macros or functions that are predefined for users to use and the compiler will handle the most of the procedures we discussed above. In AVR-GCC compiler, it provides a macro ISR() and you may use the interrupt vector names directly (predefined in the header files) as shown in Table 9.1 in the textbook. Another two macros are provided to enable and disable interrupts

Example 1

Using Timer1 so that Pin13 is toggled every .5 second while at the same time transferring data inputs from A3-A0 (PC3..0) to outputs of pin 11-8 (PB3..0). The example code is shown below.

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

void setup()
{
   DDRB |= 0x20; //make portB output
```

```
    TCCR1A = 0x43; //toggle mode, phase-correct PWM, prescaler 1024
    TCCR1B = 0x15;

    OCR1A = 0x1E84; //set tope value = 7812
    TIMSK1 = (1<<OCIE1A)|(1<<TOIE1);
    sei();          //enable global interrupt bit

    DDRC |= 0x0F;  //make PortC3..0 output
    DDRB &= ~0x0F; //make PortB3..0 input
}


void loop()
{
  char temp;
  char message[80];
  temp = PINC & 0x0F;

  //transfer data from input pins @portC3..0 to output pins @portB3..0
  PORTB &= ~0x0F;
  PORTB |= temp;

}

ISR (TIMER1_OVF_vect)
{
  PORTB ^= 0x20;
}
/*************************   End of example 1 code **********************/
```

Example 2.

Assume that INT0 and INT1 are connected to two switches named S1 and S2. Write a program that whenever S1 goes low, the contents of PORTC increases by one; and when S2 goes low, the contents of PORTC decreases by one.

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

```
void setup()
{
   DDRC |= 0x3F; //make PortC output
   EICRA = 0x0A;  //set falling edge triggered

   EIMSK = (1<<INT0)|(1<<INT1);
   sei();          //enable global interrupt bit
}


void loop()
{
  //do anything that you need
}

ISR (INT0_vect)
{
  PORTC ++;
}

ISR (INT1_vect)
{
   PORTC --;
}
/*****************************   End of example 2 code **********************/
```

References:

- ATmega328P datasheet. Available at http://www.atmel.com/Images/doc8161.pdf

Homework:

Please read chapter 9 in the textbook. You may also use ATmega328P datasheet for your reference. Problem in your textbook: 9.1-9.5.
Additional questions:

1. Which technique, interrupt or polling, avoids tying down the microcontroller?
2. True or False. The AVR programmer cannot change the memory address location assigned to the interrupt vector table.
3. To which register does the I bit belong?
4. What is the general procedure when an interrupt event occurs?

Reference Solution:

Problem in your textbook: 9.1-9.5.
9.1: Determine the appropriate bit settings for TCCR1A, TCCR1B, OCR1A and TIMSK1 using the following specific details:
- use Clear Timer on Compare (CTC) Match mode,
- disconnect both OC1 pins,
- use a pre-scale division of 1024,
- use the Output compare A Match interrupt,
- use an OCRA value such that you receive 1 interrupt per second.

A: WGM13:0 = 0100
COM1A1:0 = 00; COM1B1:0 = 00
CS12:0 = 101
So TCCR1A = 0x00; TCCR1B = 0b00001101 = 0x0D;
OCR1A = 16MHz/1024/1Hz -1 = 15624 = 0x3D08
TIMSK1 = 0b0000 0010 = 0x02 (0x03 means use both Output compare A match interrupt and overflow interrupt).

9.2. Create a function that initialize timer 1 based on the value determined in problem 9.1.

```
void Init_timer1()
{
TCCR1A = 0x00;
TCCR1B = 0x0D;
OCR1A = 0x3D08;
TIMSK1 = 0x02;
sei();
}
```

9.3.

Additional questions:

1. Which technique, interrupt or polling, avoids tying down the microcontroller?
2. True or False. The AVR programmer cannot change the memory address location assigned to the interrupt vector table.
3. To which register does the I bit belong?

What is the general procedure when an interrupt event occurs?