Interrupts
Definition of the Interrupt

- An event that requires the CPU to stop the current program execution and perform some service related to the event.

- A simple analogy
  - Reading a book and the phone rings
  - Stop reading and get the phone
  - Talk..
  - Return to the book where one read and resume to read

- The phone call is an interrupt and the talk is an **interrupt service routine (ISR)** or an **interrupt handler**.
An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt.
Polling vs. Interrupt-driven

- **Polling**
  - Actively samples the status of an external device.
  - Keep checking the port to see if the switch is being pressed.

- **Interrupt-driven programs**
  - Interrupt service routines take care of polling a device's status.
  - The main loop does not need to pay attention to the switch.
Why are interrupt used?

- Coordinate I/O activities and prevent the CPU from being tied up during data transfer process.
  - The CPU needs to know if the I/O is ready before it can proceed. Without the interrupt capability, the CPU needs to check the status of the I/O device continuously.

- Perform time-critical applications.
  - Many emergent events require the CPU to take action immediately.
  - The interrupt mechanism provides a way to force the CPU to divert from normal program execution and take immediate actions.
Interrupt Vector and Interrupt Vector Table

- Refers to the starting address of an interrupt service routine (ISR) or an Interrupt handler.
- Interrupt vectors are stored in a table called an interrupt vector table.
- The interrupt vector table must be stored in a memory location agreed upon by the microprocessor.
- The microprocessor knows how to find the vector table (and thus the ISR)
Interrupt Sequence

1. The device that requires service sets its flag bit when an event takes place.
2. The microprocessor detects that a flag is set, verifies that the corresponding enable bit is also set, and triggers an interrupt.
3. The processor status is saved automatically on the stack.
4. The microprocessor looks up the interrupt vector (the address of the ISR) for that device and puts the address into the PC.
5. The microprocessor runs the ISR.
6. At the end of the ISR, IRET must be used. IRET is a special form of return instruction which restores the processor status, so that returns to the original program.
### Table 23. Reset and Interrupt Vectors

<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Program Address(^{(2)})</th>
<th>Source</th>
<th>Interrupt Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0000^{(1)}</td>
<td>RESET</td>
<td>External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset</td>
</tr>
<tr>
<td>2</td>
<td>$0002</td>
<td>INT0</td>
<td>External Interrupt Request 0</td>
</tr>
<tr>
<td>3</td>
<td>$0004</td>
<td>INT1</td>
<td>External Interrupt Request 1</td>
</tr>
<tr>
<td>4</td>
<td>$0006</td>
<td>INT2</td>
<td>External Interrupt Request 2</td>
</tr>
<tr>
<td>5</td>
<td>$0008</td>
<td>INT3</td>
<td>External Interrupt Request 3</td>
</tr>
<tr>
<td>6</td>
<td>$000A</td>
<td>INT4</td>
<td>External Interrupt Request 4</td>
</tr>
<tr>
<td>7</td>
<td>$000C</td>
<td>INT5</td>
<td>External Interrupt Request 5</td>
</tr>
<tr>
<td>8</td>
<td>$000E</td>
<td>INT6</td>
<td>External Interrupt Request 6</td>
</tr>
<tr>
<td>9</td>
<td>$0010</td>
<td>INT7</td>
<td>External Interrupt Request 7</td>
</tr>
<tr>
<td>10</td>
<td>$0012</td>
<td>TIMER2 COMP</td>
<td>Timer/Counter2 Compare Match</td>
</tr>
<tr>
<td>11</td>
<td>$0014</td>
<td>TIMER2 OVF</td>
<td>Timer/Counter2 Overflow</td>
</tr>
<tr>
<td>12</td>
<td>$0016</td>
<td>TIMER1 CAPT</td>
<td>Timer/Counter1 Capture Event</td>
</tr>
<tr>
<td>13</td>
<td>$0018</td>
<td>TIMER1 COMPA</td>
<td>Timer/Counter1 Compare Match A</td>
</tr>
<tr>
<td>14</td>
<td>$001A</td>
<td>TIMER1 COMPB</td>
<td>Timer/Counter1 Compare Match B</td>
</tr>
<tr>
<td>15</td>
<td>$001C</td>
<td>TIMER1 OVF</td>
<td>Timer/Counter1 Overflow</td>
</tr>
<tr>
<td>16</td>
<td>$001E</td>
<td>TIMER0 COMP</td>
<td>Timer/Counter0 Compare Match</td>
</tr>
<tr>
<td>17</td>
<td>$0020</td>
<td>TIMER0 OVF</td>
<td>Timer/Counter0 Overflow</td>
</tr>
</tbody>
</table>
## Interrupt Vectors

<table>
<thead>
<tr>
<th>Vector</th>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>$0022</td>
<td>SPI, STC, SPI Serial Transfer Complete</td>
</tr>
<tr>
<td>19</td>
<td>$0024</td>
<td>USART0, RX, USART0, Rx Complete</td>
</tr>
<tr>
<td>20</td>
<td>$0026</td>
<td>USART0, UDRE, USART0 Data Register Empty</td>
</tr>
<tr>
<td>21</td>
<td>$0028</td>
<td>USART0, TX, USART0, Tx Complete</td>
</tr>
<tr>
<td>22</td>
<td>$002A</td>
<td>ADC, ADC Conversion Complete</td>
</tr>
<tr>
<td>23</td>
<td>$002C</td>
<td>EE READY, EEPROM Ready</td>
</tr>
<tr>
<td>24</td>
<td>$002E</td>
<td>ANALOG COMP, Analog Comparator</td>
</tr>
<tr>
<td>25</td>
<td>$0030(^{(3)})</td>
<td>TIMER1 COMPC, Timer/Counter1 Compare Match C</td>
</tr>
<tr>
<td>26</td>
<td>$0032(^{(3)})</td>
<td>TIMER3 CAPT, Timer/Counter3 Capture Event</td>
</tr>
<tr>
<td>27</td>
<td>$0034(^{(3)})</td>
<td>TIMER3 COMPA, Timer/Counter3 Compare Match A</td>
</tr>
<tr>
<td>28</td>
<td>$0036(^{(3)})</td>
<td>TIMER3 COMPB, Timer/Counter3 Compare Match B</td>
</tr>
<tr>
<td>29</td>
<td>$0038(^{(3)})</td>
<td>TIMER3 COMPC, Timer/Counter3 Compare Match C</td>
</tr>
<tr>
<td>30</td>
<td>$003A(^{(3)})</td>
<td>TIMER3 OVF, Timer/Counter3 Overflow</td>
</tr>
</tbody>
</table>
## Interrupt Vectors

<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Program Address(2)</th>
<th>Source</th>
<th>Interrupt Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>$003C^{(3)}</td>
<td>USART1, RX</td>
<td>USART1, Rx Complete</td>
</tr>
<tr>
<td>32</td>
<td>$003E^{(3)}</td>
<td>USART1, UDRE</td>
<td>USART1 Data Register Empty</td>
</tr>
<tr>
<td>33</td>
<td>$0040^{(3)}</td>
<td>USART1, TX</td>
<td>USART1, Tx Complete</td>
</tr>
<tr>
<td>34</td>
<td>$0042^{(3)}</td>
<td>TWI</td>
<td>Two-wire Serial Interface</td>
</tr>
<tr>
<td>35</td>
<td>$0044^{(3)}</td>
<td>SPM READY</td>
<td>Store Program Memory Ready</td>
</tr>
</tbody>
</table>

### Table 24. Reset and Interrupt Vectors Placement

<table>
<thead>
<tr>
<th>BOOTRST</th>
<th>IVSEL</th>
<th>Reset Address</th>
<th>Interrupt Vectors Start Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$0000</td>
<td>$0002</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$0000</td>
<td>Boot Reset Address + $0002</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Boot Reset Address</td>
<td>$0002</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Boot Reset Address</td>
<td>Boot Reset Address + $0002</td>
</tr>
</tbody>
</table>

Note: The Boot Reset Address is shown in Table 112 on page 284. For the BOOTRST fuse “1” means unprogrammed while “0” means programmed.
## Typical Program setup for Interrupt

<table>
<thead>
<tr>
<th>Address</th>
<th>Labels</th>
<th>Code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>jmp</td>
<td>RESET</td>
<td>; Reset Handler</td>
</tr>
<tr>
<td>$0002</td>
<td>jmp</td>
<td>EXT_INT0</td>
<td>; IRQ0 Handler</td>
</tr>
<tr>
<td>$0004</td>
<td>jmp</td>
<td>EXT_INT1</td>
<td>; IRQ1 Handler</td>
</tr>
<tr>
<td>$0006</td>
<td>jmp</td>
<td>EXT_INT2</td>
<td>; IRQ2 Handler</td>
</tr>
<tr>
<td>$0008</td>
<td>jmp</td>
<td>EXT_INT3</td>
<td>; IRQ3 Handler</td>
</tr>
<tr>
<td>$000A</td>
<td>jmp</td>
<td>EXT_INT4</td>
<td>; IRQ4 Handler</td>
</tr>
<tr>
<td>$000C</td>
<td>jmp</td>
<td>EXT_INT5</td>
<td>; IRQ5 Handler</td>
</tr>
<tr>
<td>$000E</td>
<td>jmp</td>
<td>EXT_INT6</td>
<td>; IRQ6 Handler</td>
</tr>
<tr>
<td>$0010</td>
<td>jmp</td>
<td>EXT_INT7</td>
<td>; IRQ7 Handler</td>
</tr>
<tr>
<td>$0012</td>
<td>jmp</td>
<td>TIM2_COMP</td>
<td>; Timer2 Compare Handler</td>
</tr>
<tr>
<td>$0014</td>
<td>jmp</td>
<td>TIM2_OVF</td>
<td>; Timer2 Overflow Handler</td>
</tr>
<tr>
<td>$0016</td>
<td>jmp</td>
<td>TIM1_CAPT</td>
<td>; Timer1 Capture Handler</td>
</tr>
<tr>
<td>$0018</td>
<td>jmp</td>
<td>TIM1_COMPA</td>
<td>; Timer1 CompareA Handler</td>
</tr>
<tr>
<td>$001A</td>
<td>jmp</td>
<td>TIM1_COMPB</td>
<td>; Timer1 CompareB Handler</td>
</tr>
<tr>
<td>$001C</td>
<td>jmp</td>
<td>TIM1_OVF</td>
<td>; Timer1 Overflow Handler</td>
</tr>
<tr>
<td>$001E</td>
<td>jmp</td>
<td>TIM0_COMP</td>
<td>; Timer0 Compare Handler</td>
</tr>
<tr>
<td>$0020</td>
<td>jmp</td>
<td>TIM0_OVF</td>
<td>; Timer0 Overflow Handler</td>
</tr>
</tbody>
</table>
## External Interrupts

### Table 23. Reset and Interrupt Vectors

<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Program Address(^{(2)})</th>
<th>Source</th>
<th>Interrupt Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0000(^{(1)})</td>
<td>RESET</td>
<td>External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset</td>
</tr>
<tr>
<td>2</td>
<td>$0002</td>
<td>INT0</td>
<td>External Interrupt Request 0</td>
</tr>
<tr>
<td>3</td>
<td>$0004</td>
<td>INT1</td>
<td>External Interrupt Request 1</td>
</tr>
<tr>
<td>4</td>
<td>$0006</td>
<td>INT2</td>
<td>External Interrupt Request 2</td>
</tr>
<tr>
<td>5</td>
<td>$0008</td>
<td>INT3</td>
<td>External Interrupt Request 3</td>
</tr>
<tr>
<td>6</td>
<td>$000A</td>
<td>INT4</td>
<td>External Interrupt Request 4</td>
</tr>
<tr>
<td>7</td>
<td>$000C</td>
<td>INT5</td>
<td>External Interrupt Request 5</td>
</tr>
<tr>
<td>8</td>
<td>$000E</td>
<td>INT6</td>
<td>External Interrupt Request 6</td>
</tr>
<tr>
<td>9</td>
<td>$0010</td>
<td>INT7</td>
<td>External Interrupt Request 7</td>
</tr>
</tbody>
</table>
External Interrupt Registers

- External Interrupt Control Register A - EICRA
- External Interrupt Control Register B - EICRB
- External Interrupt Mask Register - EIMSK
- External Interrupt Flag Register - EIFR
External Interrupt Control Register A - EICRA

Table 48. Interrupt Sense Control (1)

<table>
<thead>
<tr>
<th>ISCn1</th>
<th>ISCn0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The low level of INTn generates an interrupt request.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The falling edge of INTn generates asynchronously an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The rising edge of INTn generates asynchronously an interrupt request.</td>
</tr>
</tbody>
</table>

(1)
<table>
<thead>
<tr>
<th>ISCn1</th>
<th>ISCn0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The low level of INTn generates an interrupt request.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Any logical change on INTn generates an interrupt request</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The falling edge between two samples of INTn generates an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The rising edge between two samples of INTn generates an interrupt request.</td>
</tr>
</tbody>
</table>
• **Bits 7..0 – INT7 – INT0: External Interrupt Request 7 - 0 Enable**

When an INT7 – INT0 bit is written to one and the l-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Registers – EICRA and EICRB – defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.
Global Interrupt Enable

- Bit 7 – I: Global Interrupt Enable

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared in software with the SEI and CLI instructions, as described in the instruction set reference.
## Bits 7..0 – INTF7 - INTF0: External Interrupt Flags 7 - 0

When an edge or logic change on the INT7:0 pin triggers an interrupt request, INTF7:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT7:0 in EIMSK, are set (one), the MCU will jump to the 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. These flags are always cleared when INT7:0 are configured as level interrupt. Note that when entering sleep mode with the INT3:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF3:0 flags. See “Digital Input Enable and Sleep Modes” on page 70 for more information.
Example: Photo Interrupter
Polling

#include <avr/io.h>

#define GET_NOW (PINB & 1<<PD0 ? 1 : 0) // PD0가 HIGH면 1을 취함

int main(void)
{
  int now, prev;

  DDRA = 1<<PA0;       // PA0 출력, PD0 입력 설정
  PORTD |= 1<<PD0;     // PD0 내부에 풀업저항 설정

  while(1){
    for(prev = now = GET_NOW; !(now == 1 && prev == 0); now=GET_NOW) // PD0 상승[edge] 검사
    prev = now;

    PORTA ^= 1<<PA0;    // PA0 핀의 LED 반전
  }
  return 0;
}
Interrupt driven

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

volatile int req_INT0 = 1; // 최초 요청
ISR(INT0_vect)
{
  req_INT0 = 0;            // 응답 변수 기록
}

int main(void)
{
  DDRA = 1<<PA0;           // PA0 출력 방향 설정

  EIMSK |= 1<<INT0;         // INT0 인터럽트 활성화
  EICRA = 3<<ISC00;        // 상승에서 인터럽트로 설정
  PORTD |= 1<<PD0;         // PD0(INT0) 핀 내부에 풀업저항 설정
  sei();                   // 전역 인터럽트 활성화

  while(1){
    if(req_INT0 == 0){
      PORTA ^= 1<<PA0; // PA0 핀의 LED 반전
      req_INT0 = 1;   // 다시 요청함
    }
  }
  return 0;
}