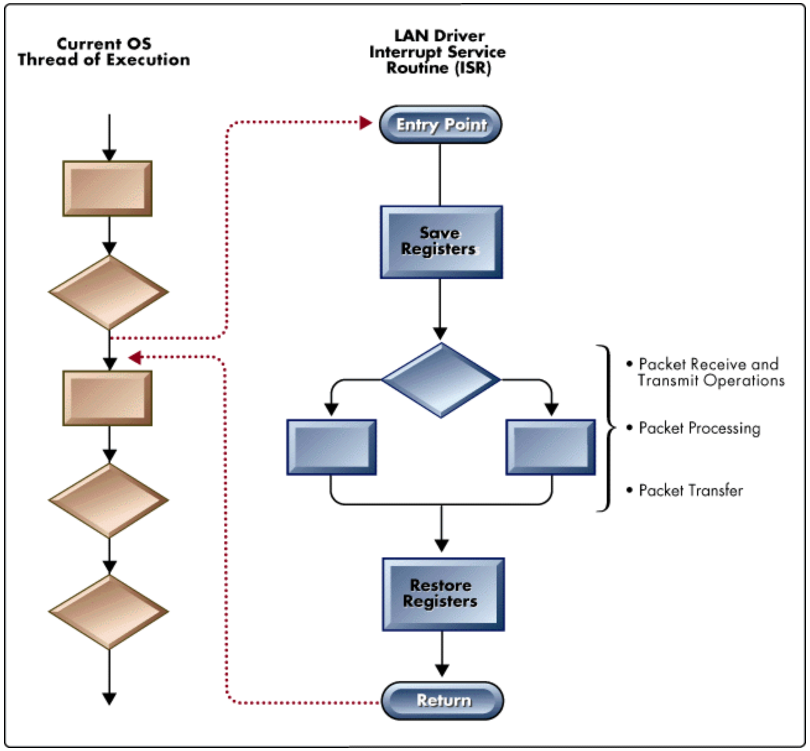
Interrupts

Definition of the Interrupt

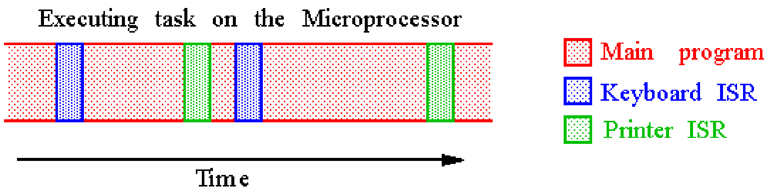
- An event that requires the CPU to stop the current program execution and perform some service related to the event.
- A simple analogy
  - Reading a book and the phone rings
  - Stop reading and get the phone
  - Talk..
  - Return to the book where one read and resume to read
- The phone call is an interrupt and the talk is an **interrupt service routine (ISR)** or an **interrupt handler**.

Interrupt Service Routine (ISR)

- An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt



http://support.novell.com/techcenter/articles/img/ana1995050101.gif



http://ece-research.unm.edu/jimp/310/slides/8086_interrupts-1.gif

Polling vs. Interrupt-driven

- Polling
  - Actively samples the status of an external devices.
  - Keep checking the port 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.

Interrupt Vectors

Table 23. Reset and Interrupt Vectors

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

Interrupt Vectors

18	$0022	SPI, STC	SPI Serial Transfer Complete
19	$0024	USART0, RX	USART0, Rx Complete
20	$0026	USART0, UDRE	USART0 Data Register Empty
21	$0028	USART0, TX	USART0, Tx Complete
22	$002A	ADC	ADC Conversion Complete
23	$002C	EE READY	EEPROM Ready
24	$002E	ANALOG COMP	Analog Comparator
25	$0030(3)	TIMER1 COMPC	Timer/Countre1 Compare Match C
26	$0032(3)	TIMER3 CAPT	Timer/Counter3 Capture Event
27	$0034(3)	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	$0036(3)	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	$0038(3)	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	$003A(3)	TIMER3 OVF	Timer/Counter3 Overflow

Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
31	$003C[3]	USART1, RX	USART1, Rx Complete
32	$003E[3]	USART1, UDRE	USART1 Data Register Empty
33	$0040[3]	USART1, TX	USART1, Tx Complete
34	$0042[3]	TWI	Two-wire Serial Interface
35	$0044[3]	SPM READY	Store Program Memory Ready

Table 24. Reset and Interrupt Vectors Placement

BOOTRST	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	$0000	$0002
1	1	$0000	Boot Reset Address + $0002
0	0	Boot Reset Address	$0002
0	1	Boot Reset Address	Boot Reset Address + $0002

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

```
Address  LabelsCode                 Comments
$0000         jmp    RESET      ; Reset Handler
$0002         jmp    EXT_INT0   ; IRQ0 Handler
$0004         jmp    EXT_INT1   ; IRQ1 Handler
$0006         jmp    EXT_INT2   ; IRQ2 Handler
$0008         jmp    EXT_INT3   ; IRQ3 Handler
$000A         jmp    EXT_INT4   ; IRQ4 Handler
$000C         jmp    EXT_INT5   ; IRQ5 Handler
$000E         jmp    EXT_INT6   ; IRQ6 Handler
$0010         jmp    EXT_INT7   ; IRQ7 Handler
$0012         jmp    TIM2_COMP  ; Timer2 Compare Handler
$0014         jmp    TIM2_OVF   ; Timer2 Overflow Handler
$0016         jmp    TIM1_CAPT  ; Timer1 Capture Handler
$0018         jmp    TIM1_COMPA; Timer1 CompareA Handler
$001A         jmp    TIM1_COMPB; Timer1 CompareB Handler
$001C         jmp    TIM1_OVF   ; Timer1 Overflow Handler
$001E         jmp    TIM0_COMP  ; Timer0 Compare Handler
$0020         jmp    TIM0_OVF   ; Timer0 Overflow Handler
```

External Interrupts

Table 23. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7

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

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

Table 48. Interrupt Sense Control[1]

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Reserved
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
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	

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
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	

- **Bits 7..0 – INT7 – INT0: External Interrupt Request 7 - 0 Enable**

When an INT7 – INT0 bit is written to one and the I-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	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
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 – I: Global Interrupt Enable**

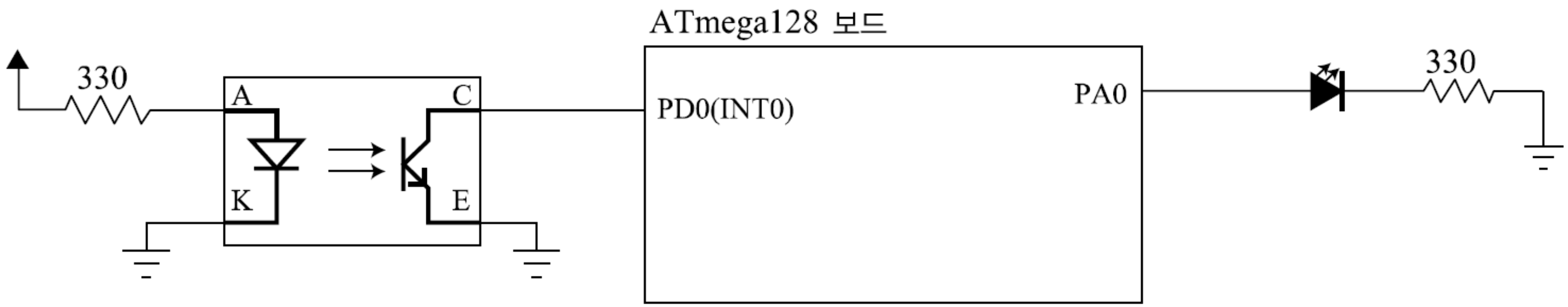
The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable 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.

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
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	

- **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 (PIND & 1<<PD0 ? 1 : 0)  // PD0가 HIGH 면 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 상승에지 검사
            prev = now;

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

Interrupt driven

```
#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;               // 상승에지 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;
```