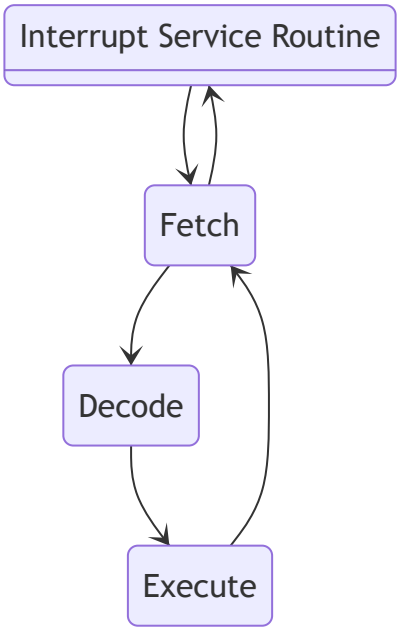
Chapter 4

4-1 Interrupt Theory

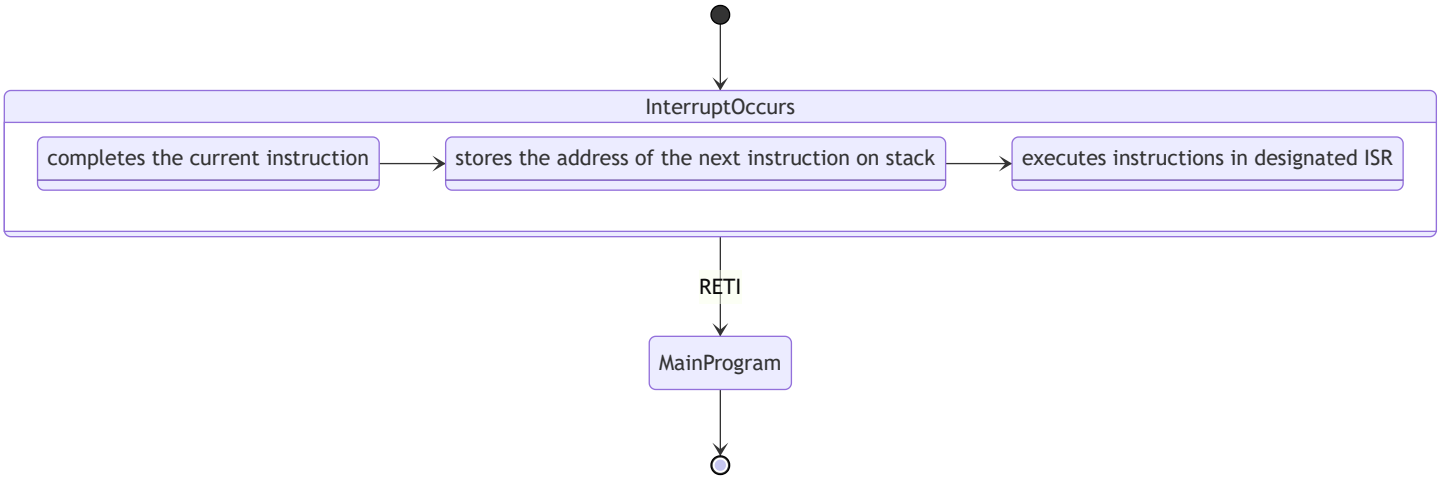
A microcontroller normally executes instructions in an orderly **fetch-decode-execute** sequence, but it must be equipped to handle unscheduled, high-priority events thant might occur inside or outside. Therefore, a microcontroller requires an interrupt system



4-2 ATmega16 Interrupt System

The ATmega16 is equipped 21 interrupt sources. **Three** originate from external interrupt sources, the remaining **eighteen** interrupts interrupts onboard.

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC	ADC Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2	External Interrupt Request 2
20	$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready



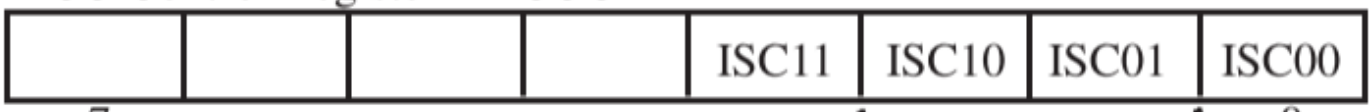
- ISR: interrupt service routine
- RETI: return from interrupt instruction

4-3 Programming An Interrupt

- Ensure the ISR for a specific interrupt is tied to the correct interrupt vector address, which points to the starting address of the ISR
- Ensure the interrupt system has been globally enabled, which is accomplished with the assembly language instruction SEI
- Ensure the specific interrupt subsystem has been locally enabled
- Ensure the registers associated with the specific interrupt have been configured correctly

External Interrupts

MCU Control Register - MCUCR



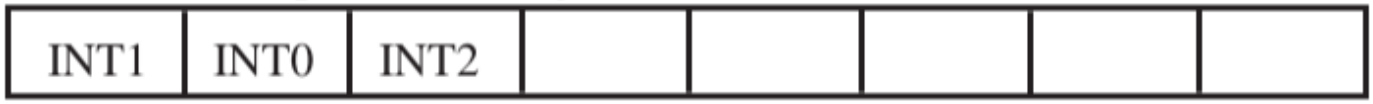
INT1
00: low level
01: logic change
10: falling edge
11: rising edge

INT0
00: low level
01: logic change
10: falling edge
11: rising edge

MCU Control and Status Register - MCUCSR

INT2
0: falling edge
1: rising edge

General Interrupt Control Register - GICR



0: interrupt disabled
1: interrupt enabled

General Interrupt Flag Register - GIFR

Notes:
- INTFx flag sets when corresponding interrupt occurs.
- INTFx flag reset by executing ISR or writing logic one to flag.

```
#pragma interrupt_handler int0_ISR:2 // interrupt handler definition

void int0_Init(void);
void int0_ISR(void);

void int0_Init(void){
    DDRD = 0xFB;    // Set PD2 as input
    PORTD &= ~0x04; // Disable pull-up resistor of PD2

    GICR = 0x40;    // Enable INT0
    MCUCR = 0x03;   // Set for rising edge

    asm("SEI");     // Enable global interrupt
}

void ISR(INT0_vect){
    /**
     * Insert interrupt specific actions here
     */
}
```

Internal Interrupts

```
#pragma interrupt_handler timer0_interrupt_isr:10

typedef unsigned char u8;
typedef unsigned int u16;

void delay(u16 number_of_interrupts);
void timer0_Init(void);

u16 input_delay;

void timer0_Init(void){
    TCCR0 = 0x04; // divide timer0 timebase by 256
    TIMSK = 0x01; // enable timer0 overflow interrupt

    asm("SEI");
}

void delay(u16 number_of_interrupts){
    TCNT = 0x00;     // reset timer0
    input_delay = 0; // reset timer0 overflow counter

    while(input_delay <=number_of_interrupts)
        ;
}

void timer0_interrupt_isr(void){
    input_delay++;
}
```