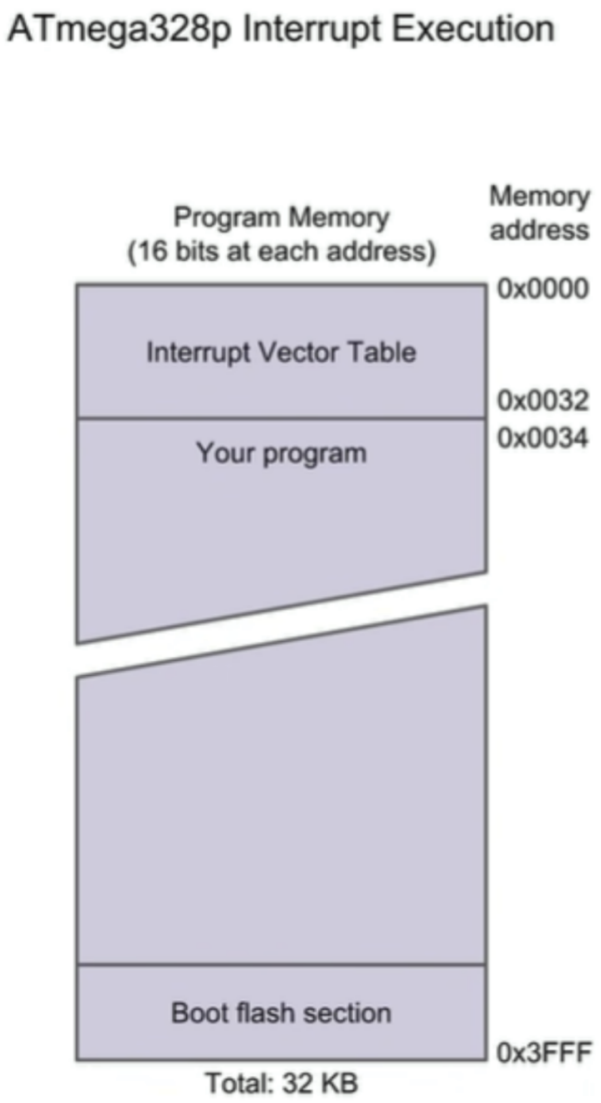
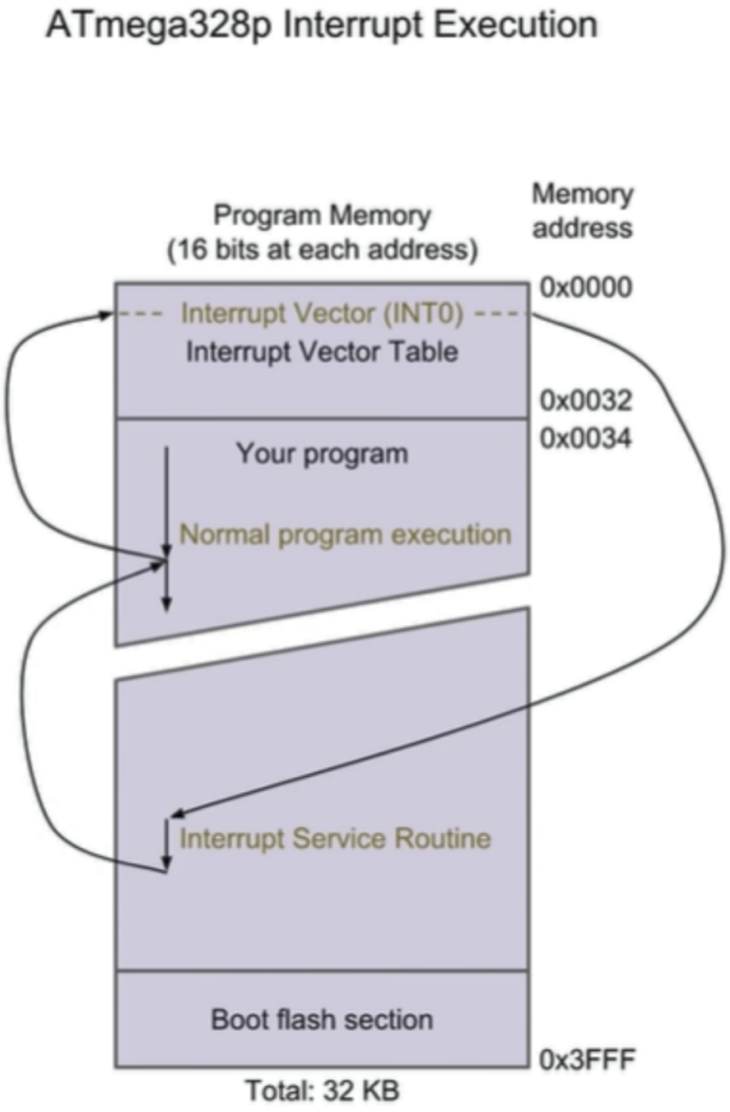
2. Interrupt Handler

On the ATMega328P microcontroller, there are three essential requirements that must met to enable an interrupt:

1. **Global Interrupt Enabled**: Interrupts must be allowed to occur globally across any part of the program. While the Arduino environment typically enables this by default, it is important to verify this when using different microcontrollers or during troubleshooting.
2. **Individual Interrupt Enabled**: Each specific interrupt must be permitted to occur via dedicated interrupt control registers.
3. **Interrupt Condition Met**: The microcontroller must receive a signal (either internal or external) that matches the predefined trigger criteria.

To enable a microcontroller to process interrupts, several steps must be followed. This module organizes these steps according to their placement in the code for easier implementation. The first step involves preparing the **Interrupt Handler** the specific code that will run when the interrupt is triggered.





Microcontrollers use a specialized memory block called the **Interrupt Vector Table**. This table stores the memory addresses of the programs to be executed for each specific type of interrupt.

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, Power-on Reset, Brown-Out Reset, 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

Vector No.	Program Address	Source	Interrupt Definition
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/Counter1 Compare Match A
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

These addresses are fixed and cannot be changed. When an interrupt occurs, the microcontroller automatically jumps to the corresponding address in the Interrupt Vector Table to execute the associated program.

For example if an external interrupt (INT0) is triggered, the microcontroller will jump to address 0x0002 to execute the code defined in that location.

See the program address between INT0 and INT1, which are 0x0002 and 0x0004 respectively. The gap between these addresses is only 2 bytes, which is the size of a single instruction in AVR assembly language. This means that the interrupt handler for INT0 must be concise and fit within this limited space, often requiring the use of a jump instruction to redirect to a larger block of code

if necessary.

Example:

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"

.org 0x0002 ; external interrupt 0 vector
    rjmp toggle_led

.global main

main:
 ; your main program here

toggle_led:
    ; main interrupt logic here
    in r16, PORTB
    eor r16, (1<<PB5)
    out PORTB, r16
    reti ; return from interrupt
```

From the above example, we have initialized the handler for the interrupt, and then created a routine named `toggle_led`, which contains the code that will be executed when the interrupt is triggered. When an INT0 occurs (for example, when a button connected to the INT0 pin is pressed), the microcontroller will jump to the `toggle_led` routine. After performing the necessary actions, we need to inform the microcontroller that the interrupt has been successfully handled and to continue with the main program. We can use the keyword `RETI`, which stands for Return from Interrupt, to achieve this.

> An interrupt will take priority over the main program, meaning that when an interrupt occurs, the microcontroller will temporarily halt the execution of the main program to execute the interrupt handler. Programmer must be cautious when writing interrupt handlers, as they should be efficient and not contain long-running code, to avoid delaying the main program for too long.

Using Vector Names

Instead of using raw addresses for interrupts, AVR assembly language provides predefined vector names that can be used to improve code readability. For example, instead of using `.org 0x0002` for the INT0 interrupt, we can use the predefined name `INT0_vect` as follows:

```
#define __SFR_OFFSET 0x00
#include "avr/io.h"

.global INT0_vect ; declare the interrupt vector as global
.global main

main:
 ; your main program here

INT0_vect: ; Toggle LED
    in r16, PORTB
    ldi r17, (1 << 5)
    eor r16, r17 ; Toggle PB5
    out PORTB, r16
    reti
```

> Pro tip: Just always use Vector Names instead of raw addresses :)

Revision #2
Created 2026-03-11 15:36:06 UTC by CH
Updated 2026-03-12 15:42:13 UTC by CH