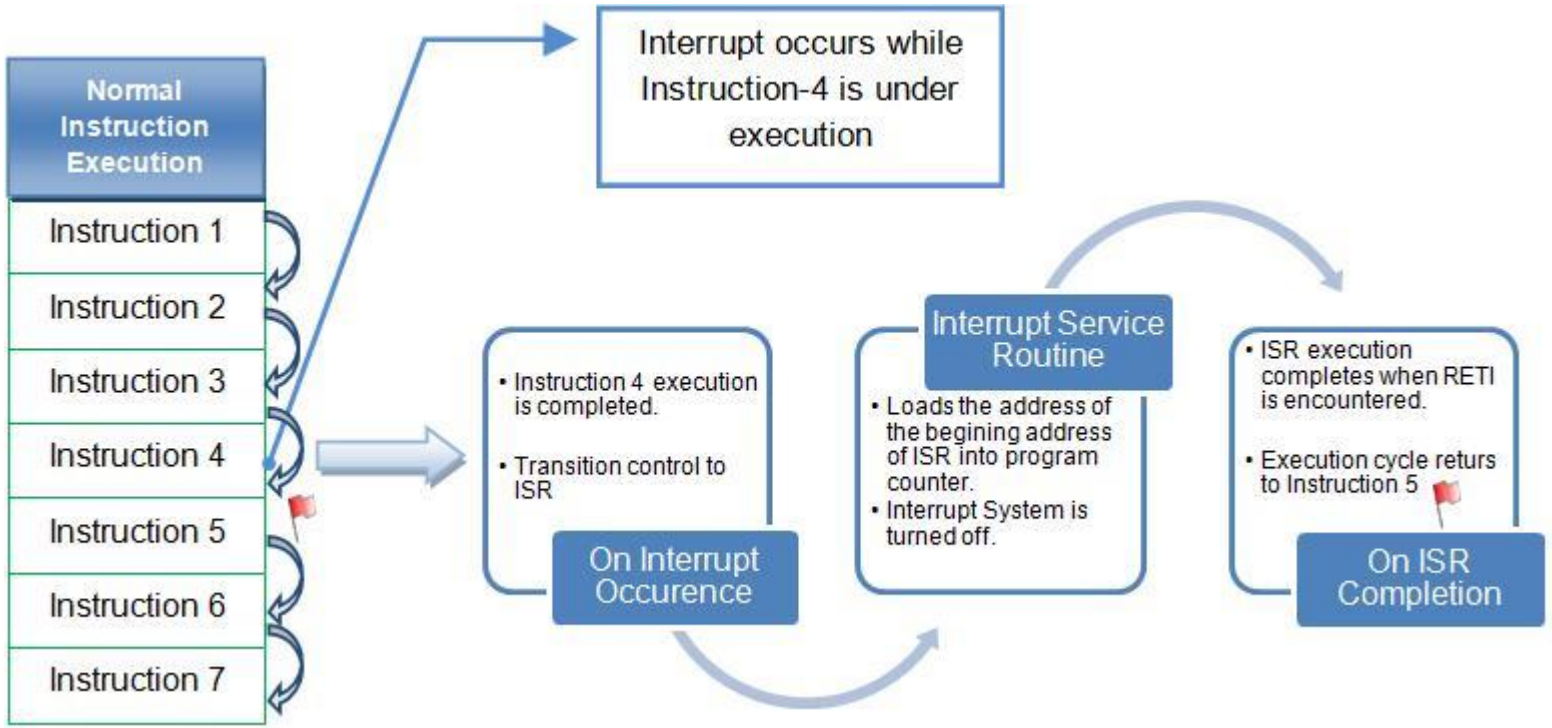
Вбудовані системи

Переривання (Interrupts)

Переривання

Переривання (Interrupts) - сигнал, що повідомляє процесору про настання деякої події. При цьому виконання поточної послідовності команд призупиняється і керування передається процедурі обробки переривання (для кожної події своя процедура обробки переривання), після чого виконання коду продовжується з того місця де він був перерваний (повернення управління). Процедура обробки переривання (Interrupt Service Routine) - це ні що інше як функція/підпрограма, яку слід виконати при виникненні певної події.



*Будемо використовувати саме слово "процедура", для того щоб підкреслити її відмінність від всіх інших функцій. Головна відмінність процедури від простих функцій полягає в тому що замість звичайного "повернення з функції" (асемблерна команда **RET**), слід використовувати "повернення з переривання" (асемблерна команда **RETI**) - "RETurn from Interrupt".*

Переривання в ATMega328

У кожного периферійного пристрою, що входить до складу AVR мікроконтролера, є як мінімум одне джерело переривання (Interrupt source). До всіх перериванъ слід зарахувати і переривання скидання - **Reset** Interrupt, призначення якого відрізняється від інших. За кожним перериванням є строго закріплений вектор (посилання), який вказує на процедуру обробки переривання (Interrupt service routine). Всі вектори перериванъ, розташовуються на самому початку пам'яті програм і разом формують "***таблицю векторів перериванъ***" (Interrupt vectors table). Кожному перериванню відповідає певний "*біт активації переривання*" (Interrupt Enable bit). Таким чином, щоб використовувати певне переривання, слід записати в його "біт активації переривання" - лог. 1. Далі, незалежно від того активували Ви чи ні переривання, мікроконтролер не почне обробку цих перериванъ, поки в "біт загального дозволу перериванъ" (Global Interrupt Enable bit в регістрі стану **SREG**) не буде записана лог. 1. Також, щоб заборонити всі переривання (на невизначений час), в біт загального дозволу перериванъ слід записати - лог. нуль.

Таблиця векторів переривань ATMega328

```
0x0000 jmp RESET       ; Reset
0x0002 jmp INT0        ; IRQ0
0x0004 jmp INT1        ; IRQ1
0x0006 jmp PCINT0      ; PCINT0
0x0008 jmp PCINT1      ; PCINT1
0x000A jmp PCINT2      ; PCINT2
0x000C jmp WDT         ; Watchdog Timeout
0x000E jmp TIM2_COMPA  ; Timer2 CompareA
0x0010 jmp TIM2_COMPB  ; Timer2 CompareB
0x0012 jmp TIM2_OVF    ; Timer2 Overflow
0x0014 jmp TIM1_CAPT   ; Timer1 Capture
0x0016 jmp TIM1_COMPA  ; Timer1 CompareA
0x0018 jmp TIM1_COMPB  ; Timer1 CompareB
0x001A jmp TIM1_OVF    ; Timer1 Overflow
0x001C jmp TIM0_COMPA  ; Timer0 CompareA
0x001E jmp TIM0_COMPB  ; Timer0 CompareB
0x0020 jmp TIM0_OVF    ; Timer0 Overflow
0x0022 jmp SPI_STC     ; SPI Transfer Complete
0x0024 jmp USART_RXC   ; USART RX Complete
0x0026 jmp USART_UDRE  ; USART UDR Empty
0x0028 jmp USART_TXC   ; USART TX Complete
0x002A jmp ADC         ; ADC Conversion Complete
0x002C jmp EE_RDY      ; EEPROM Ready
0x002E jmp ANA_COMP    ; Analog Comparator
0x0030 jmp TWI         ; 2-wire Serial interface
0x0032 jmp SPM_RDY     ; SPM Ready
```

Початок основної програми

```
; Main program start
0x0034 RESET: ldi r16, high(RAMEND)
; Set Stack Pointer to top of RAM
0x0035 out SPH, r16
0x0036 ldi r16, low(RAMEND)
0x0037 out SPL, r16
; Enable interrupts
0x0038 sei
0x003A ; ...
```

Пролог і епілог функції-обробника переривання

Набір інструкцій, які компілятор додає перед кодом обробника, називається прологом; після обробника - епілогом. Пролог функції готує регістри до використання: зберігає їх вміст в стеку; епілог відновлює регістри перед виходом, щоб перервана програма змогла продовжити роботу з ними.

```
// C code
ISR(WDT_vect)
{
// Наш код тут ...
}

; assembler code
push r1
push r0
in r0, SREG
push r0
clr r1
; Наш код тут ...
pop r0
out SREG, r0
pop r0
pop r1
reti
```

SREG - регістр статусу. Даний реєстр містить прапори, значення яких змінюються при виконанні різних команд. Наприклад, прапор нуля (Zero flag) встановлюється в одиницю, якщо в результаті виконання логічної або арифметичної операції отримано 0. Інші прапори сигналізують про переповнення, або отримання негативного результату і так далі. Зберегти вміст даного регістра при вході в обробник - золоте правило, яке дотримується і в ***AVR-GCC***. Однак **SREG** не може бути безпосередньо поміщений в стек командою push. Тому його вміст спочатку зчитується в регістр **r0**. **r0** - регістр загального призначення. ***AVR-GCC*** використовує його в якості проміжної осередки у випадках, подібних до описаного вище. Тому перед зчитуванням **SREG** вміст **r0** також поміщається в стек. r1 - регістр загального призначення, в контексті ***AVR-GCC*** використовується як нульовий регістр ("zero register") - в ньому завжди повинен бути 0. Саме тому в пролозі присутня команда очищення регістра **r1**: (clr r1) щоб бути впевненим, що в ньому міститься 0. Навіщо тоді зберігати його в стек, якщо він завжди містить 0? В тому-то й справа, що не завжди: команди множення поміщають в регістр **r1** старший байт результату. Якщо переривання виникло в той момент, коли результат множення ще не був оброблений основною програмою, то регістр **r1** може містити не нульове значення. Тому його вміст теж поміщається в стек.

Зовнішні переривання INT0, INT1 мікроконтролера ATMega328

Для того щоб дозволити обробку переривань на вході ***INT0*** необхідно встановити однойменний біт регістра **EIMSK**. Для дозволу переривань від ***INT1*** слід, відповідно, встановити біт ***INT1*** регістра. За замовчуванням (після скидання мікроконтролера) обидва біта скинуті і обробка зовнішніх переривань заборонена.

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	

Для завдання типу відслідковуваних подій на входах ***INTx*** використовується регістр **EICRA** (External Interrupt Control Register A). Значення бітів ***ISC00*** і ***ISC01*** визначають події якого типу призводять до генерації переривання **INT0**:

00 - при наявності сигналу низького рівня;
01 - при зміні сигналу від високого рівня до низького і навпаки;
10 - при зміні сигналу від високого рівня до низького;
11 - при зміні сигналу від низького рівня до високого.

Призначення бітів ***ISC10*** і ***ISC11*** аналогічні ***ISC0x*** з тією різницею, що вони визначають тип подій для переривання **INT1**.

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	

EIFR (External Interrupt Flag Register): Біт **INTF0** встановлюється в 1 при виконанні умови генерації переривання на вході **INT0** відповідно до конфігурації бітів ***ISC0x*** регістра ***EICRA***. Потім, якщо обробка переривань від **INT0** дозволена і біт **I** встановлений в 1, то буде виконаний відповідний обробник і значення біта буде скинуто. Також значення біта **INTF0** можна скинути програмно, записавши в нього значення "1". Біт **INTF1** аналогічний **INTF0**, але сигналізує про виявлення запиту переривання на вході **INT1**.

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	

Тестова програма

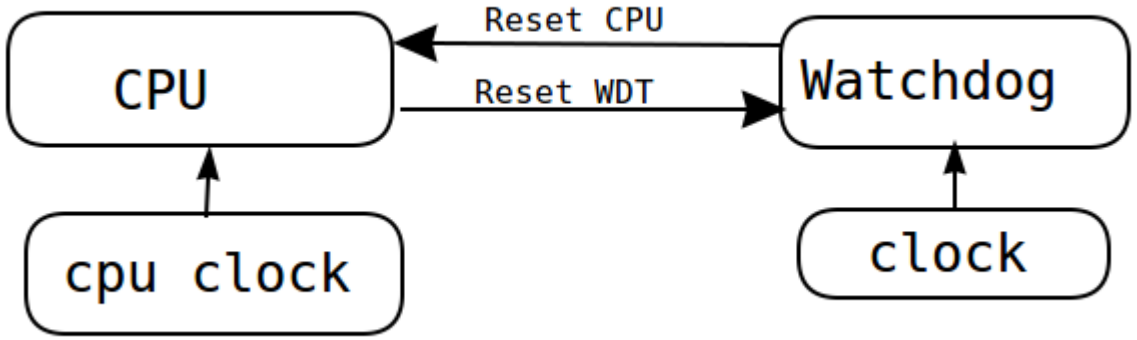
```
#include <avr/io.h>
#include <avr/interrupt.h>
#define LED_PIN PB5
#define INT_PIN PD2       // Кнопка між піном PD2 (вхід INT0) і GND
volatile uint8_t state = 0;

ISR (INT0_vect)
{
    state != state;
}

int main(void)
{
    DDRB |= 1 << LED_PIN;
    DDRD &= ~(1 << INT_PIN);
    PORTD |= 1 << INT_PIN; // підключити підтягуючий резистор до входу
    // Встановлюємо FALLING Edge на INT0
    EICRA &= ~(1 << ISC00);
    EICRA |= (1 << ISC01);
    EIMSK |= (1 << INT0);  // Дозволяємо переривання INT0
    sei();                 // Загальний дозвіл переривань

    for (;;)
    {
        if (state) PORTB |= 1 << LED_PIN;
         else PORTB &= ~(1 << LED_PIN);
    }
 return 0;
}
```

Для дозволу зовнішніх переривань і завдання режиму їх обробки необхідно виконати наступні дії:

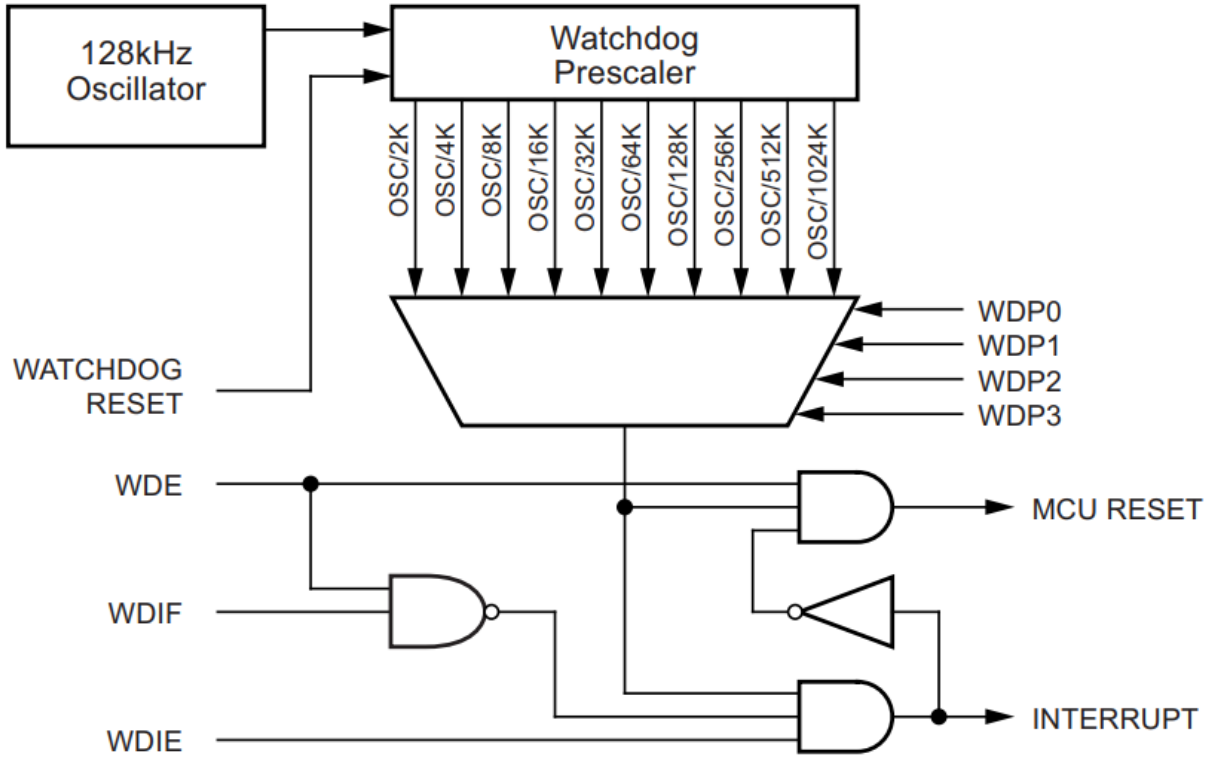
1. Задати обробник, використовуючи ключове слово ISR.
2. Визначити тип подій на вході, що генерують запит переривання (регістр EICRA).
3. Дозволити обробку зовнішнього переривання (регістр EIMSK).
4. Встановити біт I, який дозволяє обробку переривань глобально (регістр SREG).

Сторожовий таймер

Сторожовий таймер (Watchdog Timer **WDT**) - це апаратний модуль мікроконтролера який представляє собою схему контролю над зависанням системи. Даний таймер періодично скидається контрольованою системою програмно. Якщо скидання не відбулося за певний інтервал часу після попереднього скидання даного таймера, то відбувається примусове перезавантаження системи (в нашому випадку мікроконтролера).



Структурна схема сторожового таймера



Сторожовий таймер (продовження)

Регістр конфігурації сторожового таймера

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Режими роботи сторожового таймера

WDTON(1)	WDE	WDIE	Mode	Action on Time-out
1	0	0	Stopped	None
1	0	1	Interrupt mode	Interrupt
1	1	0	System reset mode	Reset
1	1	1	Interrupt and system reset mode	Interrupt, then go to system reset mode
0	x	x	System reset mode	Reset

Таблиця тайм-аутів

WDP3	WDP2	WDP1	WDP0	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	16ms
0	0	0	1	32ms
0	0	1	0	64ms
0	0	1	1	0.125s
0	1	0	0	0.25s
0	1	0	1	0.5s
0	1	1	0	1.0s
0	1	1	1	2.0s
1	0	0	0	4.0s
1	0	0	1	8.0s

WDIF (*Watchdog Interrupt Flag*) - прапор запиту на переривання. Встановлюється в 1 після закінчення встановленого інтервалу часу, коли сторожовий таймер налаштований на генерацію переривань. Прапор скидається апаратно при виконанні обробника або шляхом програмного запису в нього значення "1".

WDIE (*Watchdog Interrupt Enable*) - біт, що дозволяє обробку переривань від ***WDT***. Коли цей біт встановлений і ***WDE*** скинутий, сторожовий таймер налаштований на генерацію переривань. При встановлених ***WDIE*** і ***WDE*** сторожовий таймер спочатку буде генерувати запит переривання, скидаючи при цьому ***WDIE***, потім наступний тайм-аут таймера ініціює сигнал скидання мікроконтролера.

WDCE (*Watchdog Change Ena*ble) - біт, який дозволяє зміну біта ***WDE*** і значення дільника (***WDP[3:0]***). Для їх зміни ***WDCE*** повинен бути встановлений в "1". Після закінчення чотирьох тактів даний біт автоматично скидається, тому його слід встановлювати безпосередньо перед зміною ***WDE*** і ***WDP [3: 0]***.

WDE (*Watchdog System Reset Enable*) - даний біт дозволяє генерацію сигналу перезагрузки мікроконтролера сторожовим таймером. Його зміна контролюється бітом **WDCE**, крім того він не може бути скинутий до тих пір, поки встановлений біт ***WDRF*** регістра ***MCUSR***.

WDP [3: 0] (*Watchdog Timer Prescaler*) – біти подільника тактового сигналу ***WDT***. Допустимі комбінації бітів ***WDP [3: 0]*** і відповідні їм інтервали часу наведені в таблиці:

Сторожовий таймер (продовження)

Приклад використання сторожового таймера

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

int main(void)
{
  // Включити сторожовий таймер з
  // тайм-аутом 2 с
  wdt_enable(WDTO_2S);

    for (;;)
    {
        // Основний код програми ...
        // ...

        wdt_reset();
    }
 return 0;
}
```

Приклад використання переривань від сторожового таймера

```
#include <avr/io.h>
#include <avr/wdt.h>
volatile uint8_t wf = 0;

ISR (WDT_vect) { wf = 1; }

int main(void)
{
    cli();          // Заборона переривань на час зміни WDE і WDP
    wdt_reset();    // Скидаємо WDT
    // Дозволяємо зміну режиму та подільника WDT:
    WDTCSR |= (1 << WDCE) | (1 << WDE);
   // Встановлюємо біт WDP3 для вибору інтервалу 4с
   // і дозволяємо переривання від WDT:
    WDTCSR = (1 << WDP3) | (1 << WDIE);
    sei();          // Дозволяємо переривання
    DDRB |= 1 << PB5;

    for (;;)
    {
        if (wf != 0) {
            DDRB ^= 1 << PB5; wf = 0;
        }
    }
 return 0;
}
```