EMBEDDED

л.4
Переривання.

Палій Святослав

Український Католицький Університет
APPLIED SCIENCES FACULTY

Що таке переривання?

Переривання або **interrupt** - повідомлення процесору про настання події, яка потребує уваги щодо її обробки.

При цьому виконання поточної послідовності команд призупиняється, а керування передається **обробнику переривання** (часто називається **ISR** –**I**nterrupt **S**ervice **R**outine), який обслуговує подію, після чого керування повертається в перерваний код.

Також використовується для того щоб розбудити мікроконтролер зі сну (вийти з режиму зниженого енергоспоживання)

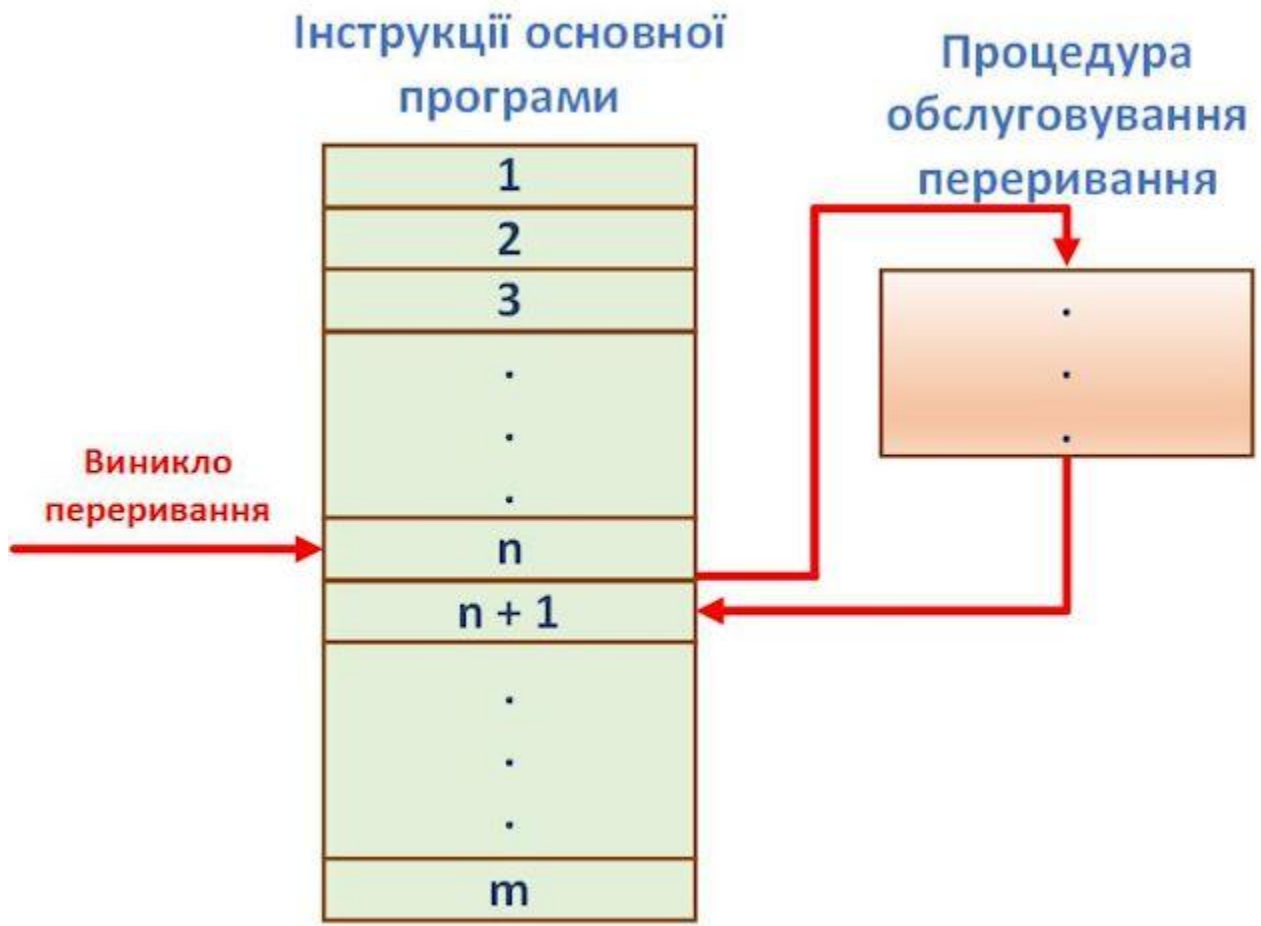
Види переривань:

- **Зовнішні (**асинхронні, апаратні) по відношенню до процесора - події, які створені зовнішніми джерелами наприклад, периферією, та можуть відбутися в довільний момент: сигнал від таймера, сигнал від I/O виводу, сигнал від комунікаційної підсистеми, чи АЦП, тощо;
- **Внутрішні** (exception) - виняткові ситуації у самому процесорі при виконанні машинного коду: наприклад ділення на нуль, звернення до неіснуючої чи неприпустимої адреси або спроба виконати неіснуючий чи недозволений код операції;
- **Програмні** - ініціюються з програмного коду виконанням спеціальної інструкції в машинному коді. Їх також деколи вважають частковим випадком внутрішніх переривань. Програмні переривання, як правило використовуються для звернення до функцій вбудованого програмного забезпечення (ROM, firmware),операційної системи, чи простого супервізора.

Приклад на Arduino

```
#include <Arduino.h>
#include <TM1637Display.h>

// Module connection pins (Digital Pins)
#define CLK 10
#define DIO 11

#define LED1 7
#define LED2 6
#define LED3 5
#define LED4 4

TM1637Display display(CLK, DIO);

void setup()
{
  pinMode(2, INPUT_PULLUP);

  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);

  display.setBrightness(0x0f);

  attachInterrupt(digitalPinToInterrupt(2), buttonIsr, FALLING);
}
```

```
void buttonIsr(int segment)
{
   digitalWrite(LED1, HIGH);
   for(int i = 0; i< 100; i++) delayMicroseconds(10000);
   digitalWrite(LED2, HIGH);
   for(int i = 0; i< 100; i++) delayMicroseconds(10000);
   digitalWrite(LED3, HIGH);
   for(int i = 0; i< 100; i++) delayMicroseconds(10000);
   digitalWrite(LED4, HIGH);
   for(int i = 0; i< 100; i++) delayMicroseconds(10000);
   digitalWrite(LED1, LOW);
   digitalWrite(LED2, LOW);
   digitalWrite(LED3, LOW);
   digitalWrite(LED4, LOW);
}

void loop()
{
  static int counter;

  display.showNumberDec(counter, false, 4, 0);
  counter = counter > 9999 ? 0 : counter + 1;
  delay(800);
}
```


Інструкції основної програми
Процедура обслуговування переривання
1
2
3
.
.
.
n
n + 1
.
.
.
m
.
.
.
Виникло переривання

Добрі практики стосовно ISR

- Обробник переривання повинен бути в розумних межах короткий та ефективний. Обробники переривань зазвичай пишуться таким чином, щоб час їх обробки був якомога меншим, оскільки під час їх роботи можуть не оброблятися інші переривання, а якщо їх буде багато (особливо від одного джерела), то вони можуть губитися.

- Обробник переривань може бути обмежений в можливості використання функцій котрі побудовані на перериваннях. Наприклад в Arduino:

    - Не можна використовувати delay() – бо ця функція використовує таймер та переривання для відліку часу

    - Функція millis() поверне коректне значення часу котре було перед викликом ISR, але при послідовних викликах в ISR значення не буде збільшуватися.

    - Функції повязані з I2C не можуть бути використані з обробника

- Деколи має зміст зробити тільки найбільш критичну частину обробки даних в ISR, а тоді поставити прапорець по якому продовжиться обробка в основному циклі програми

- Змінні котрі змінюють значення в ISR повинні бути вказані як **volatile.**

Дозвіл та заборона переривань

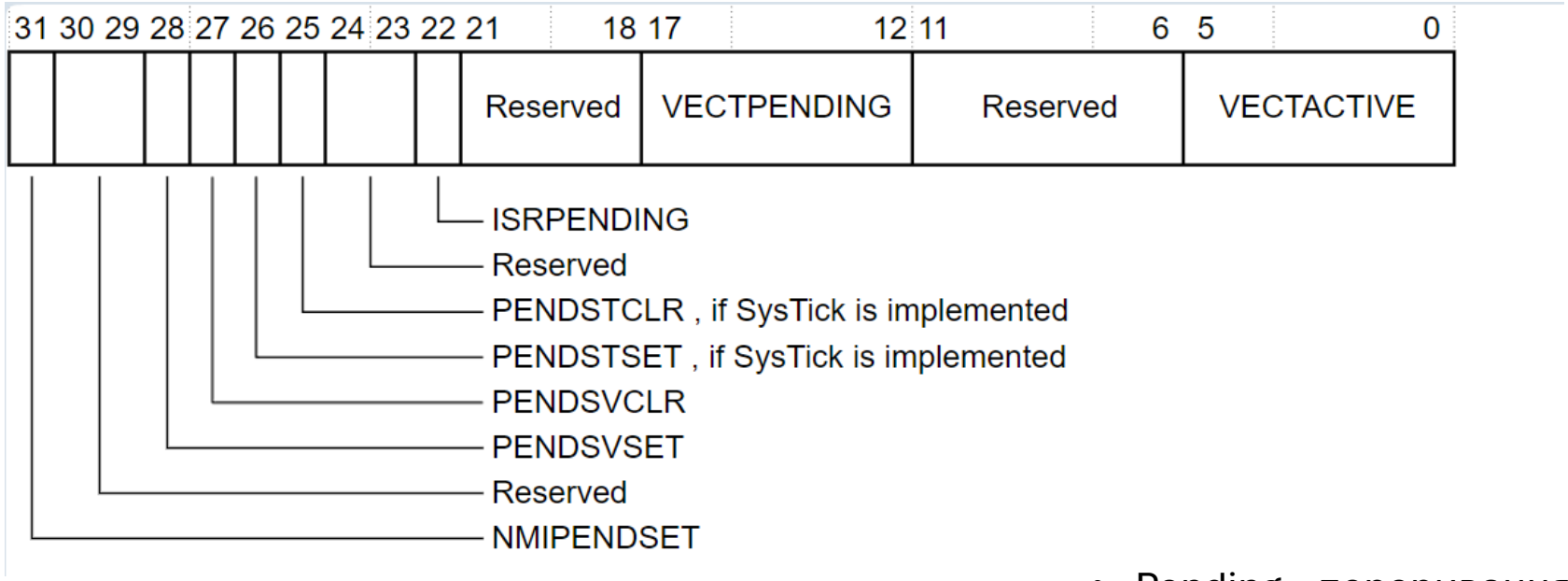
- Як правило є можливість заборонити одною командою всі переривання окрім тих які не підлягають забороні.

- Також переривання можна індивідуально маскувати, тобто забороняти виклик обробника коли переривання відбулося. Як правило (а в ARM саме так) в регістрі стану переривання відмічається що переривання сталося, але виклику ISR для маскованого переривання не відбувається.

- Не всі переривання можна маскувати. Ті, що не можна маскувати (Non maskable interrupt, NMI) — обробляються завжди, незалежно від заборон на інші переривання. Наприклад, таке переривання може бути викликане спробою доступу до неіснуючої адреси чи виконати неіснуючу команду. Наприклад для ARM Cortex-M серії не маскуються:

  - **Reset** - Це процедура, яка виконується, коли мікросхема виходить з режиму скидання.
  - **Non Maskable Interrupt (NMI)**. Якщо виникають помилки в інших обробниках винятків, буде викликано NMI. Окрім Reset, він має найвищий пріоритет серед усіх винятків.
  - **HardFault** - Це обробник для різних системних збоїв, таких як доступ до невірної пам'яті, помилки ділення на нуль та незаконні невирівняні доступи. Це єдиний обробник збоїв для архітектури ARMv6-M, але для ARMv7-M та ARMv8-M можна ввімкнути обробники збоїв з більшою деталізацією для конкретних класів помилок (наприклад, MemManage, BusFault, UsageFault).
  - **SVCall** - Обробник винятків, який викликається при виконанні інструкції Supervisor Call (svc).

Регістр котрий керує перериваннями



- Pending – переривання чекає на обробку
- Active – переривання обробляється
- Active Pending – поки оброблялося переривання знову прийшов запит на обробку.
- Inactive – нема запиту на переривання

Interrupt Set-pending Register

The ISPR forces interrupts into the pending state, and shows the interrupts that are pending. See the register summary in Table 4.2 for the register attributes.

The bit assignments are:

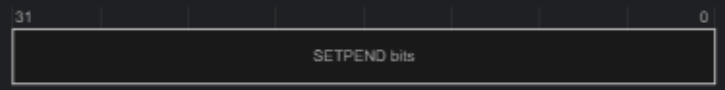


Table 4.6. ISPR bit assignments

Bits	Name	Function
[31:0]	SETPEND	Interrupt set-pending bits. Write: 0 = no effect 1 = changes interrupt state to pending.Read: 0 = interrupt is not pending 1 = interrupt is pending.

Interrupt Clear-pending Register

The ICPR removes the pending state from interrupts, and shows the interrupts that are pending. See the register summary in Table 4.2 for the register attributes.

The bit assignments are:

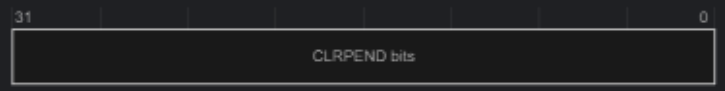


Table 4.7. ICPR bit assignments

Bits	Name	Function
[31:0]	CLRPEND	Interrupt clear-pending bits. Write: 0 = no effect 1 = removes pending state an interrupt.Read: 0 = interrupt is not pending 1 = interrupt is pending.

Interrupt Clear-enable Register

The ICER disables interrupts, and shows the interrupts that are enabled. See the register summary in Table 4.2 for the register attributes.

The bit assignments are:

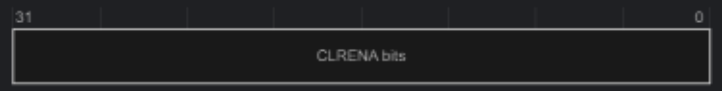
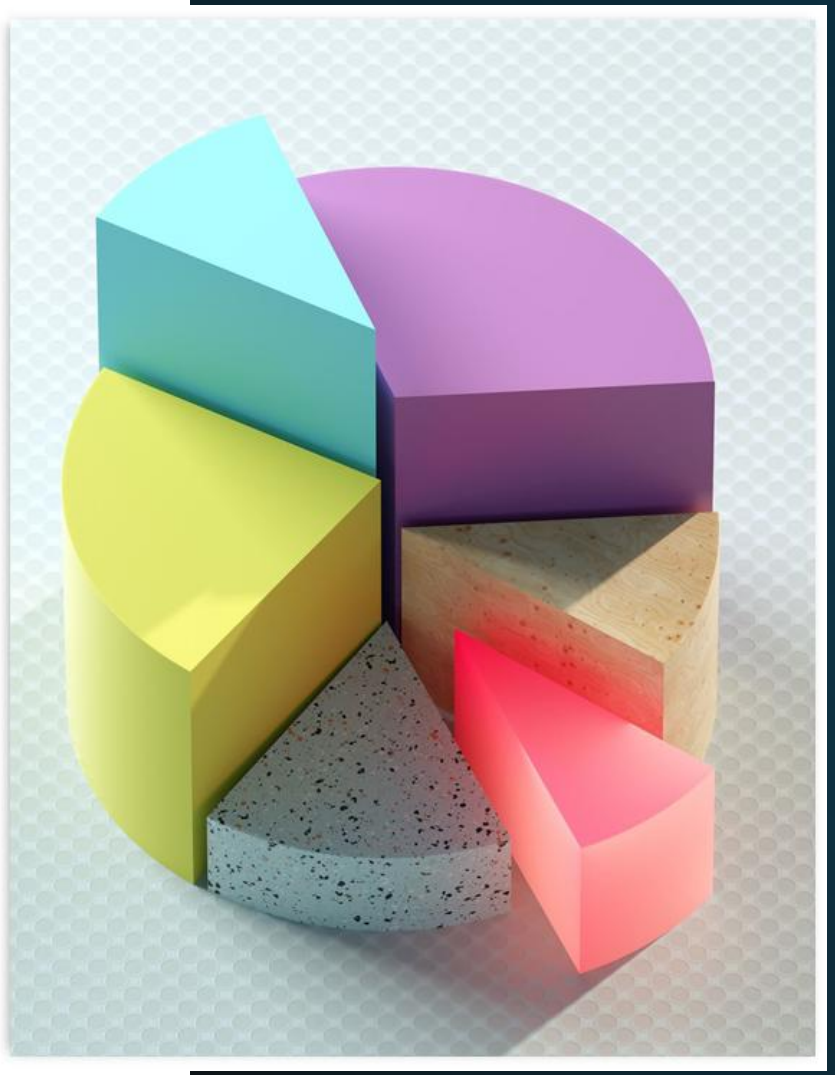


Table 4.5. ICER bit assignments

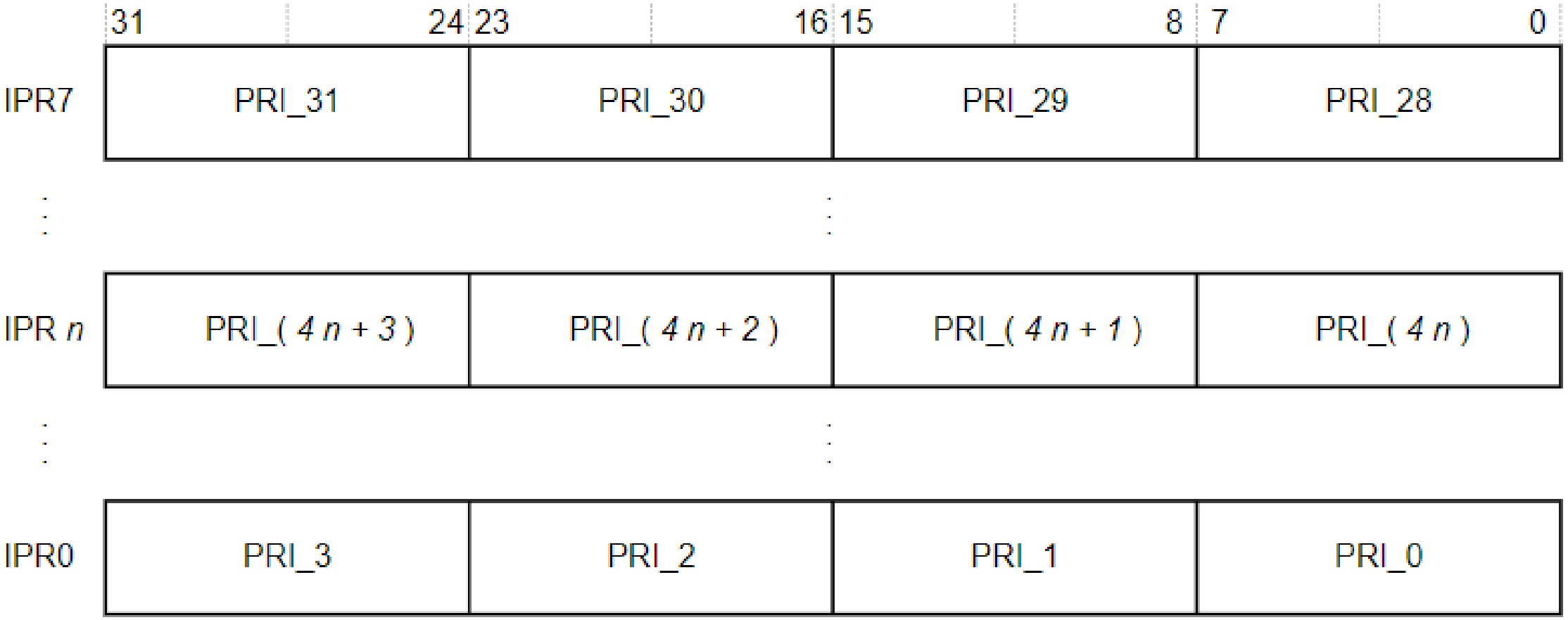
Bits	Name	Function
[31:0]	CLRENA	Interrupt clear-enable bits. Write: 0 = no effect 1 = disable interrupt. Read: 0 = interrupt disabled 1 = interrupt enabled.

NVIC – nested interrupt vector controller

- Кожне переривання має свій пріоритет від 0 до 15 (0 пріоритетніший).
- При однаковому значені пріоритету першим буде виконуватися переривання в меншим порядковим номером
- Переривання з одним і тим самим пріоритетом не переривають виконання одне одного.
- Переривання з вищим пріоритетом перериває виконання обробника переривання з нижчим пріоритетом.



- Ще почитати:
- https://developer.arm.com/documentation/dui0497/a/the-cortex-m0-processor/exception-model


31
24
23
16
15
8
7
0
IPR7
PRI_31
PRI_30
PRI_29
PRI_28
IPR n
PRI_(4 n + 3)
PRI_(4 n + 2)
PRI_(4 n + 1)
PRI_(4 n)
IPR0
PRI_3
PRI_2
PRI_1
PRI_0

NVIC programming hints

Software uses the `CPSIE i` and `CPSID i` instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void)        // Disable Interrupts
```

```
void __enable_irq(void)         // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

Table 4.9. CMSIS functions for NVIC control

CMSIS interrupt control function	Description
`void NVIC_EnableIRQ(IRQn_t IRQn)`	Enable IRQn
`void NVIC_DisableIRQ(IRQn_t IRQn)`	Disable IRQn
`uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)`	Return true (1) if IRQn is pending
`void NVIC_SetPendingIRQ (IRQn_t IRQn)`	Set IRQn pending
`void NVIC_ClearPendingIRQ (IRQn_t IRQn)`	Clear IRQn pending status
`void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)`	Set priority for IRQn
`uint32_t NVIC_GetPriority (IRQn_t IRQn)`	Read priority of IRQn
`void NVIC_SystemReset (void)`	Reset the system

Вектор переривань

таблиця переривань
vector table

Що таке вектор переривань?

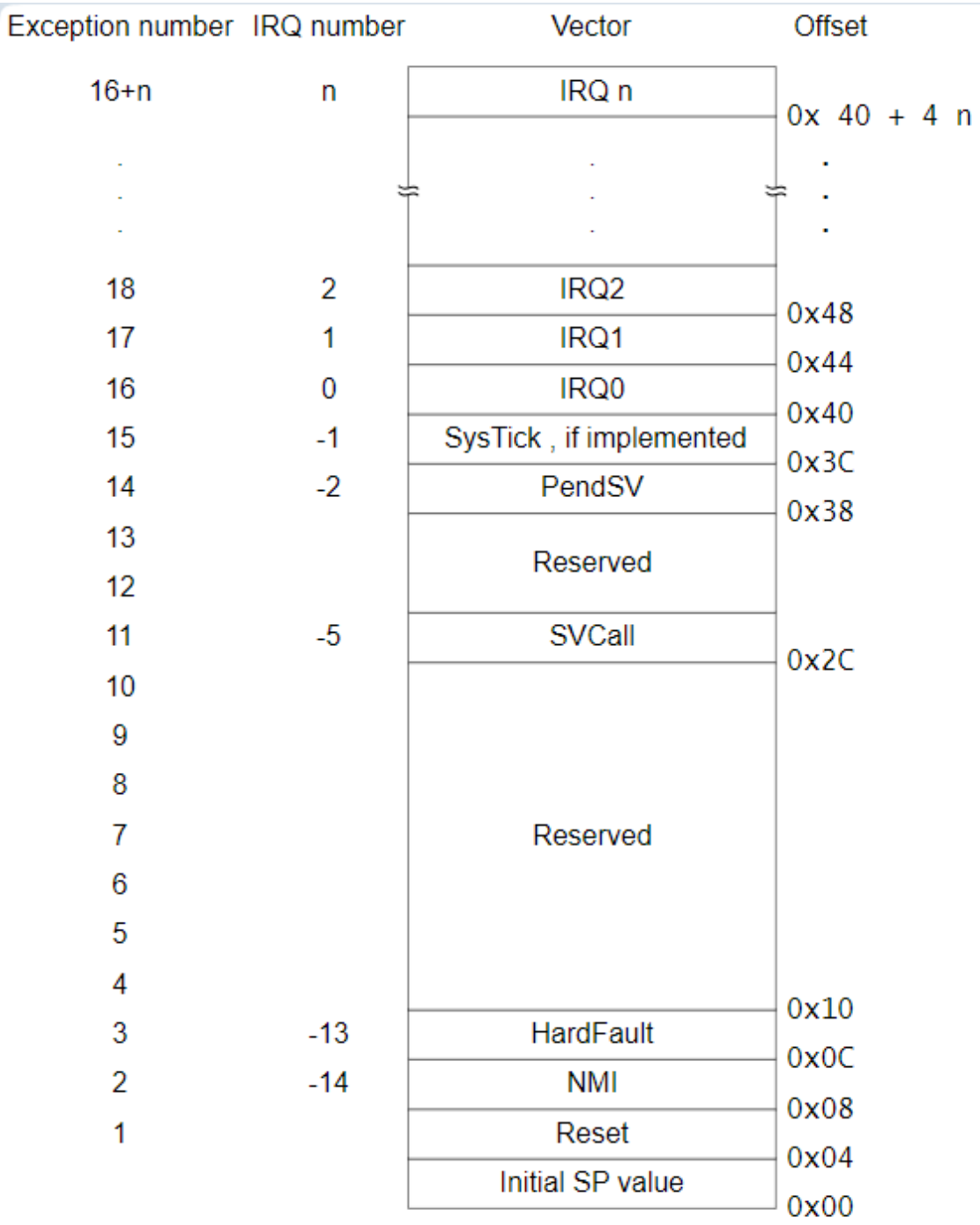
Вектор переривань — це таблиця, що містить адреси обробників для обробки подій в системі.

Роль вектора в системі

Вектор переривань забезпечує швидкий та зрозумілий доступ до обробників, коли виникає переривання в системі.

Обробка переривань

Коли виникає переривання, процесор використовує вектор для знаходження відповідного обробника.




```
.section .isr_vector
  .align 2
  .globl __isr_vector
__isr_vector:
  .long   __StackTop            /* Top of Stack */
  .long   Reset_Handler
  .long   NMI_Handler
  .long   HardFault_Handler
  .long   0                     /*Reserved */
  .long   0                     /*Reserved */
  .long   0                     /*Reserved */
  .long   0                     /*Reserved */
  .long   0                     /*Reserved */
  .long   0                     /*Reserved */
  .long   0                     /*Reserved */
  .long   SVC_Handler
  .long   0                     /*Reserved */
  .long   0                     /*Reserved */
  .long   PendSV_Handler
  .long   SysTick_Handler

   /* External Interrupts */
  .long   POWER_CLOCK_IRQHandler
  .long   RADIO_IRQHandler
  .long   UARTE0_UART0_IRQHandler
  .long   SPIM0_SPIS0_TWIM0_TWIS0_SPI0_TWI0_IRQHandler
  .long   SPIM1_SPIS1_TWIM1_TWIS1_SPI1_TWI1_IRQHandler
  .long   NFCT_IRQHandler
  .long   GPIOTE_IRQHandler
  .long   SAADC_IRQHandler
  .long   TIMER0_IRQHandler
  .long   TIMER1_IRQHandler
  .long   TIMER2_IRQHandler
  .long   RTC0_IRQHandler
  .long   TEMP_IRQHandler
  .long   RNG_IRQHandler
  [...]
```

- При запуску мікроконтролера табличка переривань розміщена в пам'яті програм (flash) починаючи з адреси 0 (для більшості MPU, включаючи ATMega, ARM Cortex-M, STM, 8051)
- В процесі виконання може бути перенесена до RAM, зазвичай також на початок RAM, але не обов'язково. Як правило перенесення таблиці відбувається в коді котрий виконується перед передачею керування до функції main(). Перенесення таблиці дає можливість призначати обробник переривання в run-time використовуючи відповідні функції, наприклад:

```
attachInterrupt(digitalPinToInterrupt(2), buttonIsr, FALLING);
```

Програмні переривання

Навіщо викликати переривання з програми?

- Зазвичай термін «програмне переривання» може бути синонімом поняттю системний виклик. Наприклад, інструкція SVC (англ. Supervisor Call) у архітектурі ARM викликає зазначену функцію у супервізора. Супервізор може виконуватися з вищим пріоритетом і мати доступ до додаткових ресурсів системи, котрі не доступні в user mode. Це необхідно для побудови як простих так і складних операційних систем, котрі дозволяють побудувати багатопоточне виконання без можливості «захоплення» процесора одним потоком. Як правило підвищення прав доступу (privileged mode) відбувається автоматично при вході в обробник SVC.
- Також завдяки механізму програмних переривань можна реалізувати доступ до system firmware – програмного коду котрий надається виробником мікроконтролера у вигляді програми в захищеній зоні пам'яті.
- Також можна просто викликати обробник переривання, який викликається по зовнішній події відповідно до логіки програмного забезпечення, наприклад ту саму дію можна запускати як по зовнішньому сигналу та і програмно.

System Call	Description	DAP Access			CPU Access
		Open	Protected	Kill	
Silicon ID	Returns the device Silicon ID, Family ID, and Revision ID	✔	✔	–	✔
Load Flash Bytes	Loads data to the page latch buffer to be programmed later into the flash row, in 1 byte granularity, for a row size of 64 bytes	✔	–	–	✔
Write Row	Erases and then programs a row of flash with data in the page latch buffer	✔	–	–	✔
Program Row	Programs a row of flash with data in the page latch buffer	✔	–	–	✔
Erase All	Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area	✔	–	–	
Checksum	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash	✔	✔	–	✔
Write Protection	This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0)	✔	✔	–	
Non-Blocking Write Row	Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Non-Blocking Program Row	Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Resume Non-Blocking	Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access	–	–	–	✔

https://www.mouser.com/pdfdocs/psoc4000_technical_reference_manual.pdf?srsltid=AfmBOorQL6VDAJANgQZLAf25B_INfXcNLOMaIhMJElTtzOJzuBVczOzA

Майже все...
але ще два нюанси

volatile

Щоб оголосити змінну як volatile, включіть ключове слово volatile перед або після типу даних у визначенні змінної. Наприклад, обидва ці оголошення оголосять беззнакову 16-бітну цілу змінну як volatile:

```
volatile uint16_t x;

uint16_t volatile y;
```

Вказівники на volatile змінні дуже поширені, особливо з регістрами вводу-виводу, що відображаються в пам'яті. Обидва ці оголошення оголосять p_reg як вказівник на volatile беззнакову 8-бітну цілу змінну:

```
volatile uint8_t * p_reg;

uint8_t volatile * p_reg;
```

Вказівники volatile на неволатильні дані дуже рідкісні (наприклад сам вказівник змінюється в ISR), але ось синтаксис:

```
uint16_t * volatile p_x;
```

І для повноти, якщо вам дійсно потрібен volatile вказівник на volatile змінну, ви напишете:

```
volatile uint16_t * volatile p_x;
```

Нарешті, якщо ви застосуєте volatile до структури або об'єднання, весь вміст структури або об'єднання буде volatile. Якщо ви не хочете такої поведінки, ви можете застосувати кваліфікатор volatile до окремих членів структури або об'єднання.

Три (і тільки три) причини для використання volatile

Змінну слід оголошувати як volatile, коли її значення може змінюватися несподівано. На практиці лише три типи змінних можуть змінюватися:

1. Регістри периферійних пристроїв, відображені в пам'яті
2. Глобальні змінні, які змінюються обробником переривань
3. Глобальні змінні, до яких звертаються кілька завдань у багатопотоковому додатку

Периферійні регістри

```
uint8_t * p_reg = (uint8_t *) 0x1234;

// чекаємо поки регістр поверне 0
do { ... } while (0 == *p_reg)
```

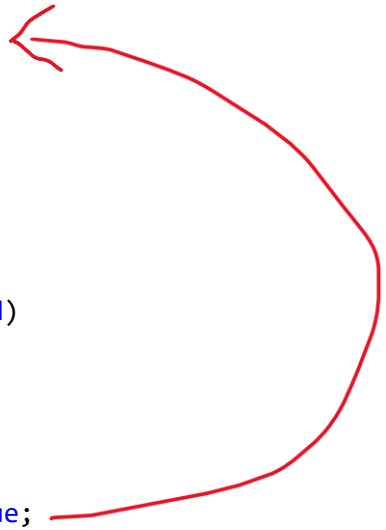
```
 mov p_reg, #0x1234
 mov a, @p_reg
loop:
 ...
 bz loop
```

```
uint8_t volatile * p_reg = (uint8_t volatile *) 0x1234;
```

```
 mov p_reg, #0x1234
loop:
 ...
 mov a, @p_reg
 bz loop
```

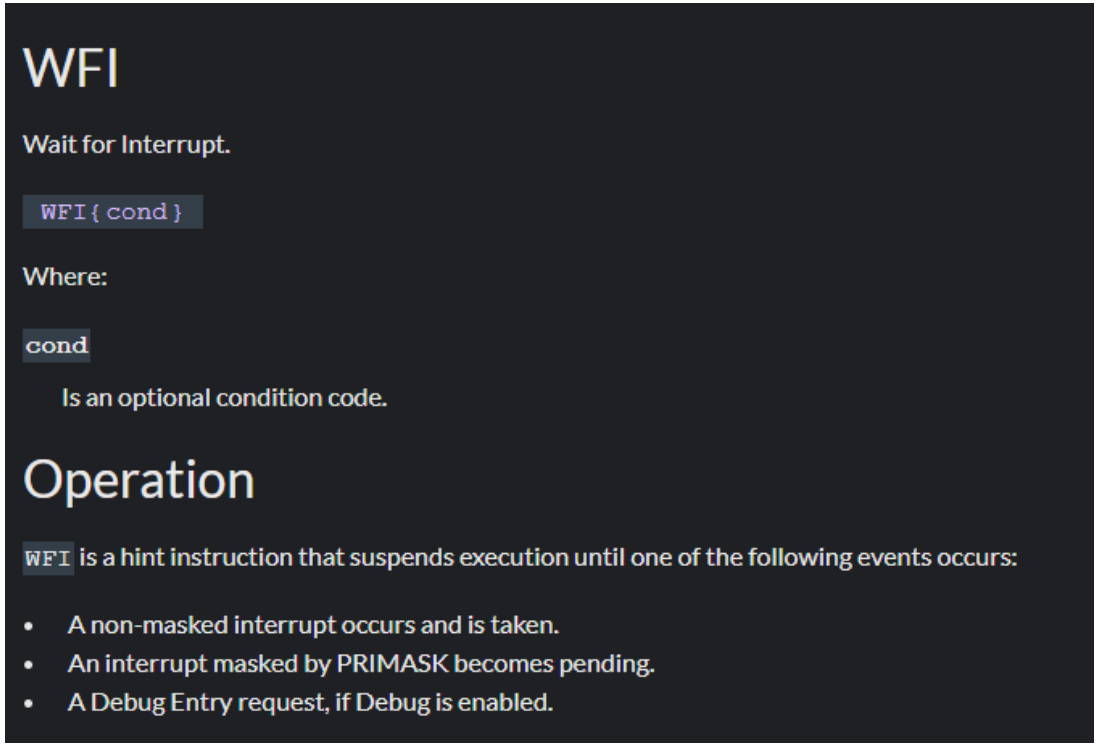
Переривання & багатопоточність

```
bool gb_etx_found = false;

void main()
{
    ...
    while (!gb_etx_found)
    {
        // Wait
    }
    ...
}

interrupt void rx_isr(void)
{
    ...
    if (ETX == rx_char)
    {
        gb_etx_found = true;
    }
    ...
}
```



Енергозбереження та переривання

В багатьох мікроконтролерах переривання є єдиним варіантом виходу з режиму глибокого енергозбереження. В режимі глибокого енергозбереження (часто його називають режимом глибокого сну, deep sleep) вимикається генератор тактових імпульсів котрий тактує мікроконтролер а також значну частину периферії, це дає можливість значно економити електроенергію, але в такому режимі мікроконтролер призупиняє виконання програми до настання якоїсь події, про яку сигналізує переривання.



WFI

Wait for Interrupt.

```
WFI{cond}
```

Where:

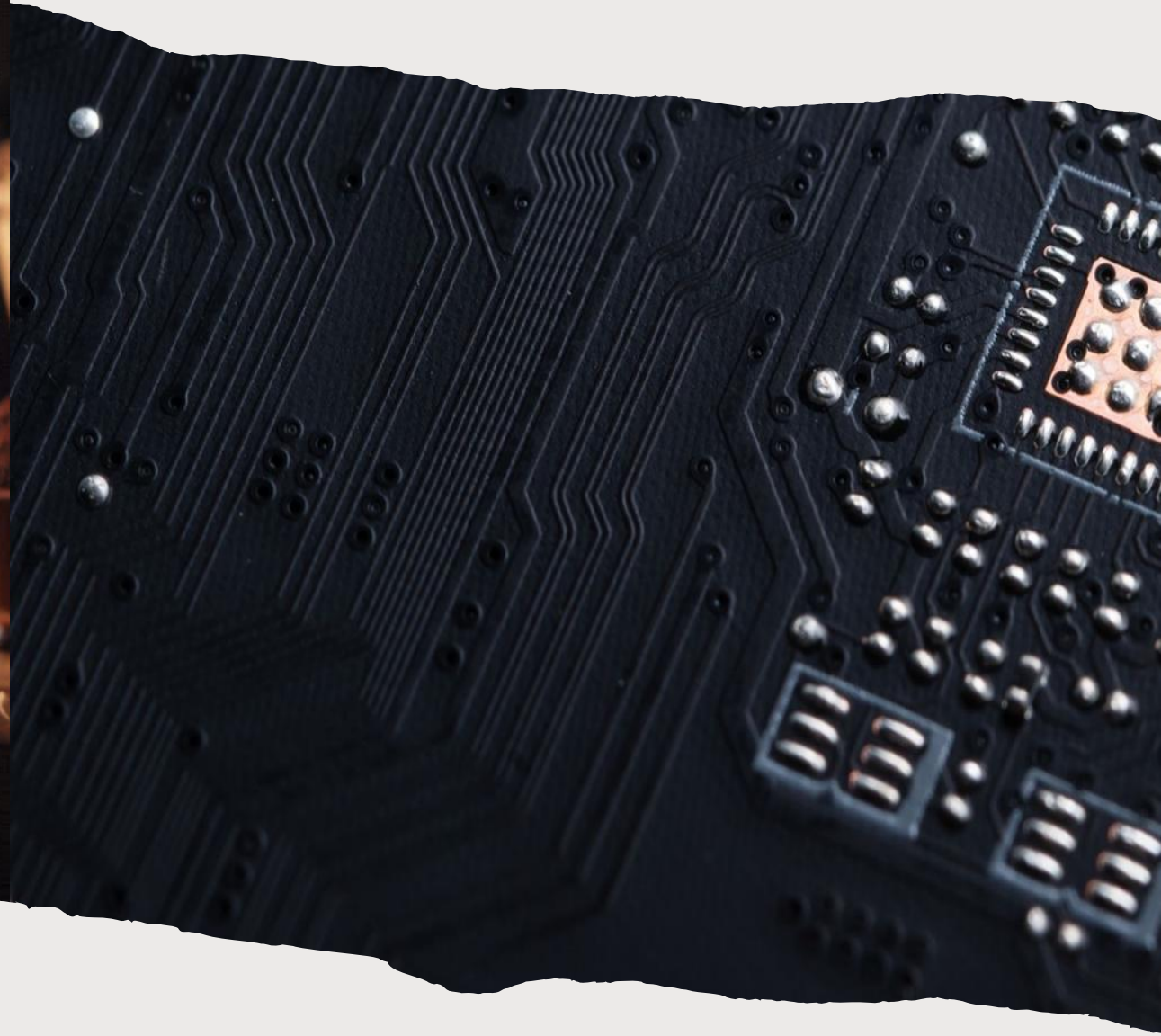
`cond`

Is an optional condition code.

Operation

`WFI` is a hint instruction that suspends execution until one of the following events occurs:

- A non-masked interrupt occurs and is taken.
- An interrupt masked by PRIMASK becomes pending.
- A Debug Entry request, if Debug is enabled.



Щиро дякую!