Systemy komputerowe: architektura i programowanie (SYKOM)

# Wykład: Przerwania obsługa sytuacji wyjątkowych

Aleksander Pruszkowski

Instytut Telekomunikacji Politechniki Warszawskiej

## PLAN WYKŁADY

- Przerwania w systemach komputerowych
- Typowe sytuacje wyjątkowe w systemach komputerowych
- Metody implementacji obsługi przerwań z przykładami w języku C/C++

# Sytuacje wyjątkowe

# Sytuacje wyjątkowe

- CPU zawiesza wykonywanie aktualnego kodu i wykonuje skok do procedur obsługi tej sytuacji
- Zdarzenia niezamierzone
  - Błąd adresowania lub rozkazu
  - Błąd argumentu
    - np.: dzielenie przez zero podczas instrukcji procesora DIV
- Zdarzenia zamierzone - wewnętrzne
  - Praca w trybie śledzenia
    - Po każdym rozkazie wykonywana jest obsługa procedury obsługi
  - Praca w trybie pułapki
    - Program działa normalnie do napotkania rozkazu pułapki (umieszczonego celowo)
  - Przerwanie programowe
    - Metoda przekazywania sterowania do systemu operacyjnego (np.: INT21)

## Sytuacje wyjątkowe

- Idea
  - Szybka reakcja na zdarzenie
- Działanie
  - Zapamiętanie stanu aktualnie wykonywanego kodu
  - Skok do kodu obsługi przerwania
  - Po wykonaniu kodu – odtworzenie stanu i powrót do przerwanego kodu
- Cel użycia
  - Obsługa w tle procesów tak aby nie utracić danych otrzymywanych z świata zewnętrznego (klawiatura, karta sieciowa, …)
  - Nisko kosztowa obsługa sytuacji sporadycznych
    - zanik napięcia zasilania nie powinien wystąpić – sprawdzanie cykliczne to tracenie zasobów CPU
- Problemy
  - Nie jawna współbieżność (np. sprzęt decyduje kiedy nastąpi wejście w obsługę)
  - Sekcja krytyczna jako remedium

- Sytuacje wyjątkowe
  - Zdarzenia zewnętrzne
    - Błąd urządzeń peryferyjnych
      - Błąd na magistrali adresowej danych (w tym parzystość czy ECC)
    - Zanik zasilania
    - Przerwania związane ze zmianą stanu sprzętu
      - Przerwanie zegarowe, przerwanie odebrania pakietu kartą sieciową, …
- Przerwania
  - Oprogramowanie może decydować czy CPU ma reagować na określone przerwania
    - Mamy dwa typy przerwań
      - Maskowalne
      - Niemaskowalne
        - W x86 przerwania niemaskowalne także można było maskować

## Sytuacje wyjątkowe

- Przerwania dlaczego w ogóle potrzebne?
  - Dane z urządzeń peryferyjnych nie muszą być dostępne natychmiast bo ich wypracowanie:
    - Wymaga czasu
      - dysk magnetyczny (HDD) musi dotrzeć (obrócić talerz, przesunąć głowicę) do żądanych informacji
    - Pewne dane wypracowuje otoczenie
      - system komputerowy może czekać na wprowadzenie danych przez użytkownika
      - dane mogą napływać z innych komputerów, np.: poprzez sieć Internet
  - Bez przerwań kod wyglądałby (tzw. polling) - oczekiwanie na naciśnięcie klawisza w PC

```
MAIN:    ...
Wait:    IN  AL, 0x64      ;odczytaj „stan klawiatury” (poprzez port 0x64)
         AND AL, 0x01      ;maska wyłączająca inne przyczyny zmiany stanu
         JZ  Wait          ;nie naciśnieto klawisza – skacz do etykiety Wait
         IN  AL, 0x60      ;odczytaj kod naciśnietego klawisza
         ...               ;obróbka otrzymanego numeru klawisza
```

## Sytuacje wyjątkowe

- Przerwania dlaczego w ogóle potrzebne, cd.?
  - Podejście poprawne, ale nie efektywne
    - Program nie robi nic poza sprawdzaniem stanu określonego portu (instrukcja IN)
  - Z obsługą przerwania napisalibyśmy

```
MAIN:     ...
          CALL Enable_IRQ_KEYBOARD ; włącz generowanie przerwań przez klawiaturę
          ...
IRQ_KEY:  PUSH AL
          IN   AL, 0x64    ;odczytaj „stan klawiatury” (poprzez port 0x64)
          AND  AL, 0x01    ;maska wyłączająca inne przyczyny zmiany stanu
          JNZ  SKIPIRQ     ;nie naciśnieto klawisza – opuść przerwanie
          IN   AL, 0x60    ;odczytaj kod naciśnietego klawisza
          ...              ;obróbka otrzymanego numeru klawisza
SKIPIRQ:  POP AL
          IRET             ;powrót z przerwania
```

  - W powyższym w czasie działania programu głównego przerwanie będzie w tle obsługiwać klawiaturę – oszczędzamy moc obliczeniową
    - Koszt: komplikacja oprogramowania
      - Konieczność buforowania danych – w przerwaniu nie przetwarzamy danych!
      - Program główny i kod obsługi przerwania mogą działać współbieżnie – wymagana synchronizacja danych

## Sytuacje wyjątkowe

- Przerwania - Jak CPU obsługuje przerwania

1)pod koniec fazy wykonania operacji maszynowych sprawdzany jest stan wejść

- INT - przerwania sprzętowe (ich może być wiele)
- NMI – przerwania niemaskowalne (są CPU nie posiadające tego typu wejścia)

2)jeżeli stan nie wymaga wejścia w procedurę obsługi przerwania - CPU przechodzi do fazy pobrania

3)zawartość rejestrów EPC, EFLAGS odkładana jest na stos

4)generowany jest adres pod który wykonany będzie skok i wpisywany jest on do EPC, i przechodzi CPU do fazy pobrania

- Taki sposób realizacji jest optymalny - przerwanie nie dzieli żadnej z faz pracy CPU

## Sytuacje wyjątkowe

- Przerwania – Co programista musi zrobić w kodzie obsługi przerwania

1)Jeżeli to możliwe wyłączyć przerwania

- niekontrolowana „rekurencja przerwań” może skończyć się katastrofą (!)
  - przepełnienie stosu może występować w „dziwnych momentach”, trudnych w diagnostyce
    - decydujące mogą być: kolejność wywołań przerwań, stopień zapisania stosu
- niektóre procesory domyślnie wyłączają możliwość przerywania wykonywanych właśnie przerwań
  - jeżeli programista życzy sobie - może odblokować możliwość przyjmowania przerwań

2)Zapamiętać na stosie wszystkie rejestry używane w funkcji obsługi przerwania

- PUSH EAX; PUSH EBX; … lub prościej (mniej efektywnie) PUSHA
- czasami wcześniej trzeba przełączyć stos

3)Wykonać zadania możliwie ***najkrócej jak to możliwe!***

4)Odtworzyć w odwrotnej kolejności ze stosu zawartości rejestrów

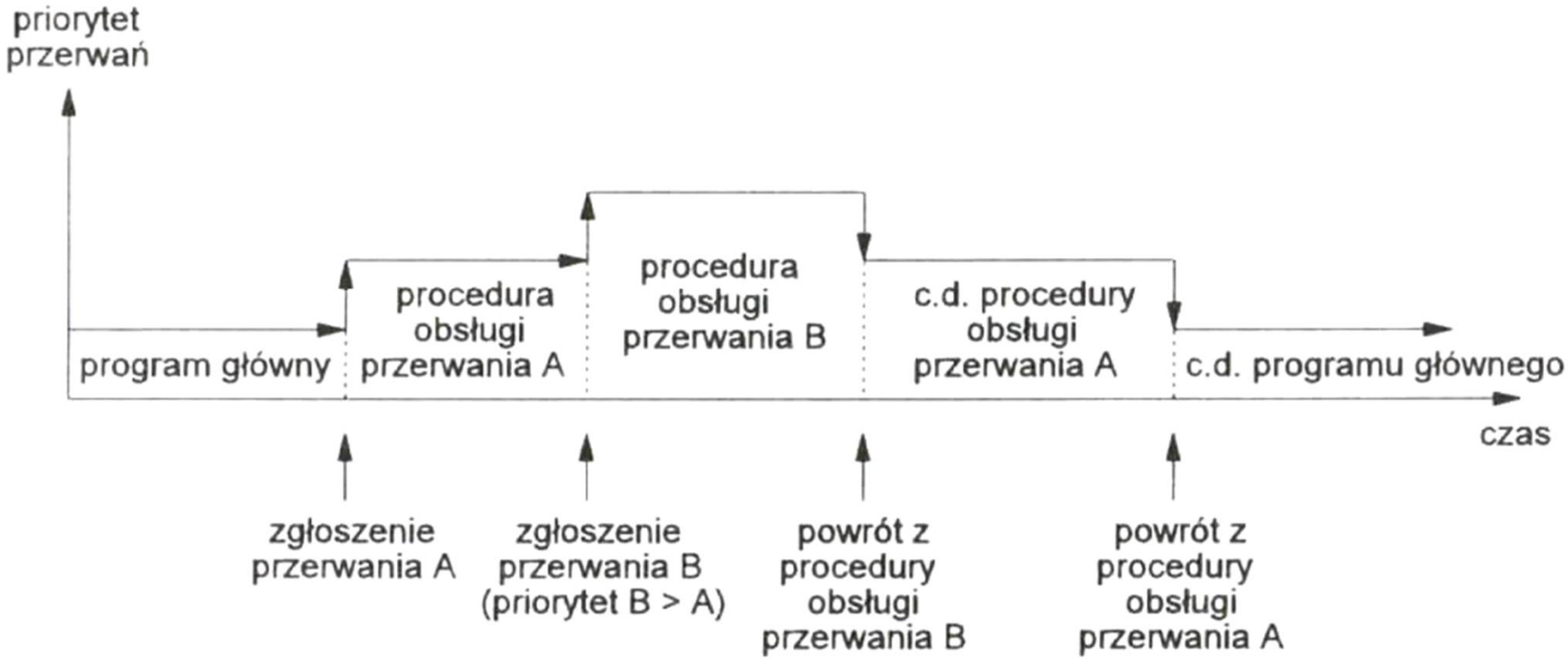
- POP …; POP EBX; POP EAX lub prościej POPA

5)Wykonać instrukcję IRET / RETI

- w innych procesorach zwykłą instrukcję RET lub jej odpowiednik

# Sytuacje wyjątkowe

- Priorytety przerwań
  - Przerwania przerywające inne przerwania
    - „rekurencja przerwań”




Źródło: Tomasz STARECKI, „Mikrokontrolery jednoukładowe rodziny 51”

## Sytuacje wyjątkowe

- Przerwania – w x86
  - W trybie rzeczywistym tablica przerwań umiejscowiona jest pod adresem 0
  - Struktura - jeden wpis zajmuje 4B i jest adresem kodu procedury obsługi przerwań
    - wybrane przykładowe lokacje (nie adresy) wpisów w tablicy przerwań x86
      - 0x00 Dzielenie stałoprzecinkowe przez zero
      - 0x01 Przerwanie programowe (gdy TF=1 przerwanie wywoływane po każdym rozkazie)
      - 0x02 Przerwanie NMI związane z błędem parzystości pamięci lub koprocesora
      - 0x03 Pułapka programowa (generowana przez INT 3)
      - 0x06 Niedozwolony kod rozkazu
      - 0x0B Segment nieobecny w pamięci głównej
      - 0x0C/0x0D Przekroczenie segmentu stosu
      - 0x0E Błąd strony
  - W trybie chronionym przerwania opisuje tablica deskryptorów przerwań IDT
    - umiejscowienie tej pamięci wyznacza rejestr IDTR

- Przerwania – w x86
  - Wybrane przerwania sprzętowe w x86 (obsługiwane przez układ **8259**)
    - IRQ0 Zegar systemowy (lokacja w tablicy przerwań: 0x08)
    - IRQ1 Obsługa klawiatury (0x09)
    - IRQ2 Kaskada dla podrzędnego kontrolera IRQ (0x0A)
    - IRQ3/IRQ4 Obsługa portu szeregowego COM2/COM1 (0x0B, 0x0C)
    - IRQ6 Obsługa sterownika napędu dysków elastycznych (0x0E)
    - IRQ5/IRQ7 Obsługa portu równoległego LPT2/LPT1 (0x0D, 0x0F)
    - IRQ8 Obsługa zegara czasu rzeczywistego (0x70)
    - IRQ9 Wywołanie przerwania IRQ2 gdy połączone kaskadowe (0x71)
    - IRQ14 Obsługa sterownika twardego dysku (0x76)
    - Nie przypisane: IRQ10, IRQ11, IRQ12, IRQ15
      - Karty rozszerzeń mogły z nich korzystać

# Sytuacje wyjątkowe

- Tablica wektorów przerwań innych procesorów – 80C51
  - Wycinek obok przedstawia listę przerwań z nazwami i adresami
  - Procedury obsługi przerwań rozmieszczone co 8B
    - w tym MCU zakłada się, że są przerwania których kod może zmieścić się w 8B
    - przykład procedury obsługi zmieniającej stan bitu 1 portu 3 np.:

```
ORG 000BH ;gdzie umieścić kod
          ;tu pobudzenie to T0
CPL P3.1  ;P3.1=!P3.1
RETI      ;powrót z przerwania
```

Źródło: Tomasz STARECKI, „Mikrokontrolery jednoukładowe rodziny 51”

Tabela 4.1. Zestawienie przerwań występujących w wybranych mikrokontrolerach rodziny 51

| Źródło przerwania | Wskaźniki przerwania | Wektor przerwania | Uwagi |
|---|---|---|---|
| przerwanie zewnętrzne INT0 | IE0 | 0003H | |
| przerwanie zewnętrzne INT1 | IE1 | 0013H | za wyjątkiem CE558 i CE559 |
| przerwanie zewnętrzne INT1 i licznik sekundowy | IE1+SECINT | 0013H | CE558, CE559 |
| przerwanie zewnętrzne INT2 | IQ2 | 003BH | mikrokontrolery odmiany CL |
| przerwanie zewnętrzne INT2 | IEX2 | 004BH | C515, C515A, C517, C517A |
| przerwanie zewnętrzne INT2 | IE2 | 0053H | C51GB |
| przerwanie zewnętrzne INT2 | IE2 | 0043H | C310, C320, C323, C520, C530 |
| przerwanie zewnętrzne INT3 | IQ3 | 0043H | mikrokontrolery odmiany CL |
| przerwanie zewnętrzne INT3 | IEX3 | 0053H | C515, C515A, C517, C517A |
| przerwanie zewnętrzne INT3 | IE3 | 005BH | C51GB |
| przerwanie zewnętrzne INT3 | IE3 | 004BH | C310, C320, C323, C520, C530 |
| przerwanie zewnętrzne INT4 | IQ4 | 004BH | mikrokontrolery odmiany CL |
| przerwanie zewnętrzne INT4 | IEX4 | 005BH | C515, C515A, C517, C517A |
| przerwanie zewnętrzne INT4 | IE4 | 0063H | C51GB |
| przerwanie zewnętrzne INT4 | IE4 | 0053H | C310, C320, C323, C520, C530 |
| przerwanie zewnętrzne INT5 | IQ5 | 0053H | mikrokontrolery odmiany CL |
| przerwanie zewnętrzne INT5 | IEX5 | 0063H | C515, C515A, C517, C517A |
| przerwanie zewnętrzne INT5 | IE5 | 006BH | C51GB |
| przerwanie zewnętrzne INT5 | IE5 | 005BH | C310, C320, C323, C520, C530 |
| przerwanie zewnętrzne INT6 | IQ6 | 005BH | mikrokontrolery odmiany CL |
| przerwanie zewnętrzne INT6 | IEX6 | 006BH | C515, C515A, C517, C517A |
| przerwanie zewnętrzne INT6 | IE6 | 0073H | C51GB |
| przerwanie zewnętrzne INT7 | IQ7 | 0063H | mikrokontrolery odmiany CL |
| przerwanie zewnętrzne INT8 | IQ8 | 006BH | mikrokontrolery odmiany CL |
| przerwanie zewnętrzne INT9 | IQ9 | 0073H | odmiana CL, za wyjątkiem CL580 |
| układ licznikowy T0 | TF0 | 000BH | za wyjątkiem C748–C752 |

## Sytuacje wyjątkowe

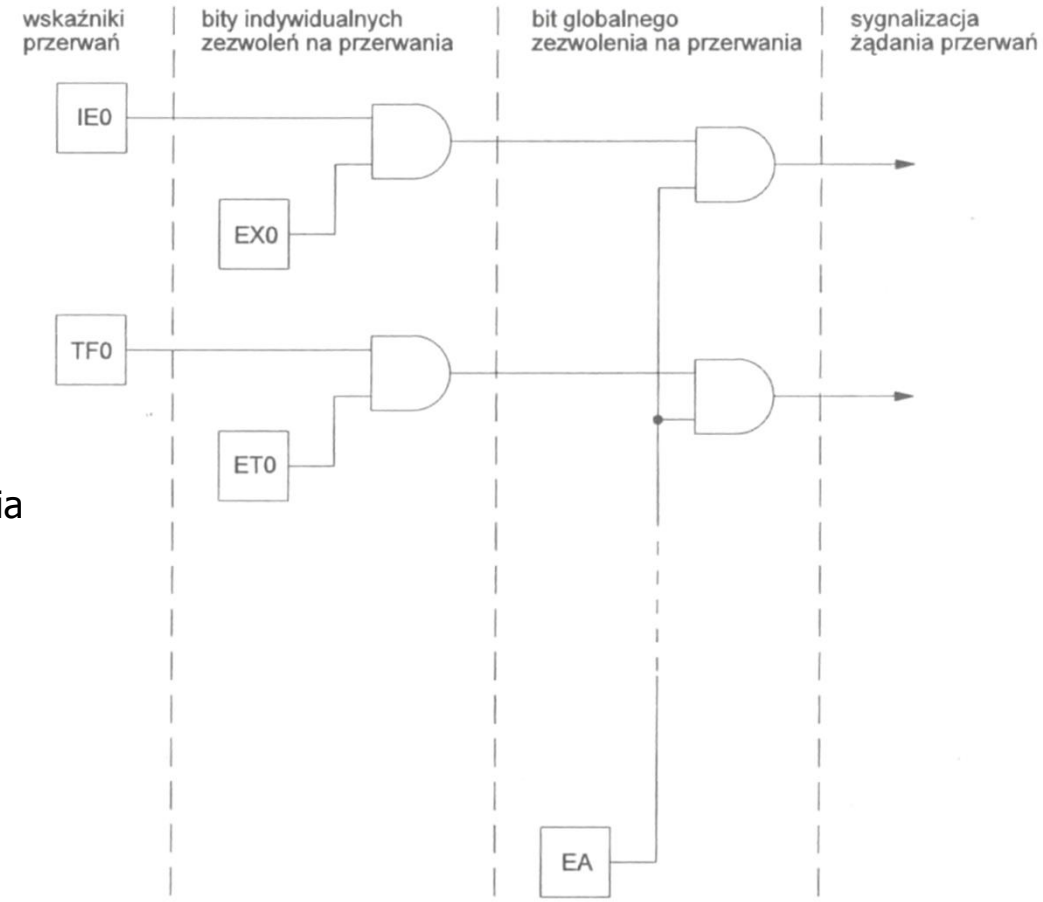
- 80C51 – kompilator SDCC
  - Sposób zapisu funkcji obsługi przerwań

```
void timer_isr (void) __interrupt (1) __using (1){
   ...
}
```

  - Specjalne słowa kluczowe pozwalają odpowiednio
    - ustalić numer wektora przerwań „__interrupt (numer_przerwania)”
    - ustalić bank rejestrów używanych przez przerwanie „__using (numer_banku)”
      - metoda szybkiego przełączenia kontekstu – eliminuje konieczność zapamiętywania rejestrów na stosie

## Sytuacje wyjątkowe

- Obsługa przerwań w 80C51 (wycinek)
  - Włączenie przerwań wymaga ustawienia wielu tzw. bitów sterujących
    - `sbit` w strukturze SFR
    - na rysunku
      - EA – włącza wszystkie przerwania
      - EX0, ET0 – włącza przerwania od określonych źródeł
      - TF0 – przepełnienie licznika jako źródło przerwania




Rys. 4.1. Fragment struktury układu przerwań przedstawiający sposób funkcjonowania wskaźników przerwań i bitów zezwoleń na przerwania.

Źródło: Tomasz STARECKI, „Mikrokontrolery jednoukładowe rodziny 51”, Warszawa 1996

## Sytuacje wyjątkowe

- Tablica wektorów przerwań innych procesorów – AVR
  - Jeden wpis w tablicy mieści 4B (opcode są 2B)
    - Tablica to spis instrukcji JMP <ADDRES_WLASCIWEGO_HANDLERA>
      - JMP z argumentem tu zajmuje 4B

| Vector No. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset |
| 2 | 0x0002 | INT0 | External Interrupt Request 0 |
| 3 | 0x0004 | INT1 | External Interrupt Request 1 |
| 4 | 0x0006 | INT2 | External Interrupt Request 2 |
| 5 | 0x0008 | INT3 | External Interrupt Request 3 |
| 6 | 0x000A | INT4 | External Interrupt Request 4 |
| 7 | 0x000C | INT5 | External Interrupt Request 5 |
| 8 | 0x000E | INT6 | External Interrupt Request 6 |
| 9 | 0x0010 | INT7 | External Interrupt Request 7 |
| 10 | 0x0012 | TIMER2 COMP | Timer/Counter2 Compare Match |
| 11 | 0x0014 | TIMER2 OVF | Timer/Counter2 Overflow |
| 12 | 0x0016 | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 13 | 0x0018 | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 14 | 0x001A | TIMER1 COMPB | Timer/Counter1 Compare Match B |
| 15 | 0x001C | TIMER1 OVF | Timer/Counter1 Overflow |
| 16 | 0x001E | TIMER0 COMP | Timer/Counter0 Compare Match |

Źródło: 8-bit AVR Microcontroller ATmega128A DATASHEET COMPLETE

## Sytuacje wyjątkowe

- Tablica wektorów przerwań innych procesorów – AVR, cd.
  - Przykład kodu startowego dla procesora AVR

```
$0000          jmp RESET       ; Reset Handler
$0002          jmp EXT_INT0    ; IRQ0 Handler
$0004          jmp EXT_INT1    ; IRQ1 Handler
$0006          jmp EXT_INT2    ; IRQ2 Handler
$0042          jmp TWI    ; Two-wire Serial Interface Interrupt Handler
$0044          jmp SPM_RDY     ; SPM Ready Handler (re-programowanie
...                            ; pamięci FLASH)
...
$0046 RESET:   ldi r16, high(RAMEND) ; Main program start
$0047          out SPH,r16           ; Set stack pointer to top of RAM
$0048          ldi r16, low(RAMEND)
$0049          out SPL,r16
$004A          sei             ; Enable interrupts
$004B          <instr> xxx   ; Main code
...
```

**RAMEND** – określa ile pamięci posada dany typ AVR MCU
W AVR po położeniu czegoś na stosie wskaźnik stosu SP jest zmniejszany(!)

# Sytuacje wyjątkowe

- Przerwania w AVR-GCC

```
#include <avr/interrupt.h>
#include <util/atomic.h>

volatile char my_flag=0;

ISR(INT0_vect) {
  my_flag=1; //ustawienie naszej flagi
  ... //... i jak najmniej innych operacji
}

int main(void) {
  ...         //Konfiguracja przerwań
  char local_copy_my_flag;
  for(;;){
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
      local_copy_my_flag=my_flag;
    }
    if(local_copy_my_flag!=0){
      ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
        my_flag=0;          //atomowe skasowanie flagi
      }
      ...  //tu wiemy że przerwanie zostało wywołane, możemy wykonywać dłuższe
      ...  //operacje i przetważać informacje/dane „złapane” w przerwaniu
    }
  }
}
```

**ATOMIC_BLOCK** – tworzy blok kodu będącą sekcją atomową

**ATOMIC_RESTORESTATE** – przywrócenie stanu systemu z przed wejściem do sekcji atomowej (tu rej. SREG)

**ISR** – funkcja obsługi przerwania

**volatile** – zmienna nie będzie podlegać optymalizacji (przydatne gdy kod funkcji przerwania i kod główny korzystają z takich zmiennych)

**INT0_vect** – wektor przerwania wspierany przez dane MCU (np.: wejście INT0 czyli „External Interrupt 0”)

## Sytuacje wyjątkowe

- Przerwania w Arduino

```
const byte ledPin = 13;
const byte interruptPin = 2;  //numer wejścia przerwania

volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

**digitalPinToInterrupt** – zamiana numeru pinu w notacji Arduino na notacje wymaganą przez m.in. attachInterupt

**attachInterupt** – podłącz funkcję „blink()” pod przerwanie związane ze zmianą stanu pinu interruptPin (2), nie wszystkie piny potrafią „generować” przerwania

- W tym przykładzie brak sekcji atomowej, aby pokazany kod był prostszy

Baza: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

## Sytuacje wyjątkowe

- Tablica wektorów przerwań mikrokontrolerów zgodnych z ARM – STM32F0xxx Cortex-M0
  - Obszar wektorów przerwań od IRQ podzielono na wpisy o wielkości 4B
    - Numery IRQ ujemne dla zaznaczenia, że są to wyjątki (exceptions) a nie przerwania sprzętowe

| Exception number(1) | IRQ number(1) | Exception type | Priority | Vector address or offset(2) | Activation |
|---|---|---|---|---|---|
| 1 | - | Reset | -3, the highest | 0x00000004 | Asynchronous |
| 2 | -14 | NMI | -2 | 0x00000008 | Asynchronous |
| 3 | -13 | Hard fault | -1 | 0x0000000C | Synchronous |
| 4-10 | - | Reserved | - | - | - |
| 11 | -5 | SVCall | Configurable(3) | 0x0000002C | Synchronous |
| 12-13 | - | Reserved | - | - | - |
| 14 | -2 | PendSV | Configurable(3) | 0x00000038 | Asynchronous |
| 15 | -1 | SysTick | Configurable(3) | 0x0000003C | Asynchronous |
| 16 - 47 | 0 - 31 | Interrupt (IRQ) | Configurable(3) | 0x00000040 and above(4) | Asynchronous |

Źródło: STM32F0xxx Cortex-M0 programming manual

## Sytuacje wyjątkowe

- Deklarowanie funkcji obsługi przerwań w GCC (bez wsparcia specjalistycznych bibliotek)
  - dla CPU ARM:

```
void f() __attribute__ ((interrupt ("IRQ"))); //lub: FIQ, SWI, ABORT, UNDEF, ...
void f(void) {
  ...
}
```

  - Dla CPU RISC-V:

```
void f(void) __attribute__ ((interrupt ("user")));//lub: ”supervisor”,”machine”
void f(void) {
  ...
}
```

  - Dla X86:

```
struct interrupt_frame;            //definicja zgodna z manualem CPU
__attribute__((interrupt)) void f(struct interrupt_frame* frame){
...
}
```

# Sytuacje wyjątkowe

- Deklarowanie funkcji obsługi przerwań dla STM32 w GCC (ze wsparciem specjalistycznych bibliotek)

```
void NMI_Handler(void) __attribute__ ((weak, alias("Default_Handler")));
void HardFault_Handler(void) __attribute__ ((weak, alias("Default_Handler")));
void MemManage_Handler(void) __attribute__ ((weak, alias("Default_Handler")));
...
void CRYP_IRQHandler(void) __attribute__ ((weak, alias("Default_Handler")));
void HASH_RNG_IRQHandler(void) __attribute__ ((weak, alias("Default_Handler")));
void FPU_IRQHandler(void) __attribute__ ((weak, alias("Default_Handler")));
...
void Reset_Handler(void){
  ...
  __libc_init_array();
  main();
}
uint32_t vectors[] __attribute__((section(".isr_vector")))={
  STACK_START,
  (uint32_t)Reset_Handler,
  (uint32_t)NMI_Handler,
  (uint32_t)HardFault_Handler,
  (uint32_t)MemManage_Handler,
   ...
  (uint32_t)CRYP_IRQHandler,
  (uint32_t)HASH_RNG_IRQHandler,
  (uint32_t)FPU_IRQHandler
};
```

A co to jest „Default_Handler”?

Funkcja „pułapka”:

```
void Default_Handler(void){
  while(1);
}
```

Podstawiana - gdy w kodzie nie będzie implementacji właściwej funkcji np.: FPU_IRQHandler(), …

A jak sprawić by ta tablica „wylądowała” we właściwym miejscu?

Skrypt linkera: stm32_ls.ld, opisuje sekcje ”.text” a w niej jest instrukcja aby na jej początku wstawić treść: ”*(.isr_vector)”

Źródło: https://github.com/niekiran/baremetalembedded/blob/master/source/stm32_startup.c

**Dziękujemy za uwagę!**