

Akademia
Górnictwo-Hutnicza
w Krakowie
Katedra Elektroniki



Laboratorium mikrokontrolerów

Ćwiczenie 7

Przerwania

Autor: Paweł Russek
Tłumaczenie: Sebastian Koryciak
<http://www.fpga.agh.edu.pl/tm>
ver. 8.06.15

1. Wprowadzenie

1.1. Cel ćwiczenia

Głównym celem ćwiczenia jest przedstawienie koncepcji przerwań oraz ich programowania dla mikrokontrolerów z rodziny AVR. Na początku laboratorium zostaną przedstawione główne ich zalety. Następnie zaprezentowane zostaną dostępne źródła przerwań dla mikrokontrolera ATmega32. Później omówimy rejestry i obsługę przerwań. Szczególną uwagę zwrócono na przerwanie od timerów oraz od zdarzeń zewnętrznych. W trakcie ćwiczeń student wykona proste zadania weryfikujące nabytą wiedzę i umiejętności.

1.2. Konieczne wiadomości wstępne

Zaliczenie ćwiczeń laboratoryjnych 1, 2, 3, 4, 5, oraz 6.

1.3. Przygotowanie płytki ZL2AVR

2. Podstawy przerwań

2.1. Przerwania a odpytywanie

Do tej pory sami musieliśmy wykonywać testy dotyczące konkretnych wydarzeń. Przykładowo aby rozstrzygnąć czy przycisk jest wciśnięty sprawdzaliśmy odpowiedni bit w rejestrze PinX. Metodę tą nazywamy odpytywaniem (ang. polling). W trakcie tej procedury mikrokontroler w trybie ciągłym monitoruje status danego urządzenia, na przykład przycisku. Odpytywanie nie jest metodą optymalną, ponieważ marnuje większość dostępnego przez procesor czasu oraz niepotrzebnie zużywa energię. Podejściem alternatywnym jest procedura przerwań. W tej metodzie urządzenia, które wymagają obsługi przez mikrokontroler informują go o tym po przez aktywowanie sygnału przerwania. Mikrokontroler po otrzymaniu takiego sygnału zatrzymuje wykonywanie rutynowych operacji i uruchamia program związany z danym przerwaniem. Program taki nazywany jest procedurą obsługi przerwania (ang. interrupt service routine (ISR)) lub "interrupt handler", czyli obsługą przerwania. W przeciwieństwie do metody odpytywania, procedura przerwań pozwala na zastosowanie priorytetów. Tym sposobem, jeżeli w danym momencie wystąpi więcej niż jedno przerwanie, jako pierwsze zostanie obsłużone o wyższym priorytecie.

2.2. Tok postępowania przy wykonywaniu przerwania

Mikrokontroler po aktywowaniu przerwania:

1. dokończa aktualnie wykonywaną instrukcję i zapisuje adres kolejnej instrukcji na stosie,
2. skacze do tablicy wektorów przerwań, która go kieruje pod adres wybranej procedury obsługi przerwania,
3. wykonuje całą procedurą obsługi przerwania, aż do ostatniej instrukcji, którą jest wyjście z obsługi przerwania (RETI).
4. Po wykonaniu instrukcji RETI mikrokontroler wraca do miejsca, w którym mu

przerwano. Przywraca ze stosu licznik programu (PC) i od tego adresu wykonuje kolejne instrukcje.

Uwaga

Stos spełnia bardzo krytyczną rolę w trakcie obsługi przerwania. Podczas wykonywania procedury obsługi przerwania należy zwrócić szczególną uwagę aby ilość instrukcji *pop* i *push* była taka sama. Należy również przypilnować aby każdy rejestr wykorzystywany podczas ISR był na samym początku obsługi zrzucany na stos, a pod koniec przywrócony, tak aby nie zakłócić pracy przerwanych programów.

3. Źródła przerw dla AVR

Istnieje szereg wydarzeń, które mogą automatycznie ostrzec nas, gdy tylko nastąpią. Każde źródło przerwania musi mieć swoją procedurę obsługi przerwania. W procesorach AVR dla każdego przerwania przewidziana jest lokacja w pamięci, która przechowuje adres programu ISR. Cała przestrzeń w pamięci zawierająca adresy procedur obsługi przerwania nazywana jest tablicą wektorów przerw (ang. interrupt vector table).

Poniżej znajduje się fragment tablicy wektorów przerw dla ATmega32.

Rodzaj przerwania	lokalizacja w EEPROM	etykieta .equ
Reset	\$0000	
External Interrupt request 0	\$0002	INT0addr
External Interrupt request 1	\$0004	INT1addr
External Interrupt request 2	\$0006	INT2addr
Timer/Counter2 Compare Match	\$0008	OC2addr
Timer/Counter2 Overflow	\$000A	OVF2addr
Timer/Counter1 Capture Event	\$000C	ICP1addr
Timer/Counter1 Compare Match A	\$000E	OC1Aaddr
Timer/Counter1 Compare Match B	\$0010	OC1Baddr
Timer/Counter1 Overflow	\$0012	OVF1addr
Timer/Counter0 Compare Match	\$0014	OC0addr
Timer/Counter0 Overflow	\$0016	OVF0addr

Kod poniżej przedstawia typowe i najczęściej spotykane ustawienie dla obsługi Resetu i wybranych adresów wektorów przerw dla ATmega32. Zawarta w nim jest obsługa następujących przerw: *External Interrupt request 0*, *Timer/Counter0 Compare Match*, oraz *Timer/Counter0 Overflow*.

Obsługa innych w danym momencie wymaganych przerw również może zostać dodana zgodnie z nazwami w powyższej tabeli.

Example.

```
.include "m32def.inc"

.org 0
    jmp reset

.org INT0addr                ; External Interrupt Request 0
    jmp external_interrupt_0

.org OC0addr                 ; Timer/Counter0 Compare Match
    jmp timer0_compare_match

.org OVF0addr                ; Timer/Counter0 Overflow
    jmp timer0_overflow

reset:
    ldi r16,high(RAMEND)      ; Main program start
    out SPH,r16              ; Set Stack Pointer to top of RAM
    ldi r16,low(RAMEND)
    out SPL,r16
    ;... some code

external_interrupt_0:
    ;... some code
    reti

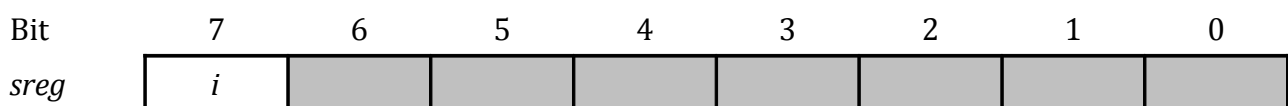
timer0_compare_match
    ;... some code
    reti

timer0_overflow:
    ;... some code
    reti
```

4. Globalne włączanie i wyłączanie przerwań

Wszystkie przerwania mogą zostać dezaktywowane (wyłączone). Oznacza to, że mikrokontroler na nie nie zareaguje. Przerwania muszą być aktywowane (włączone) w oprogramowaniu. Bit D7 w rejestrze statusowym (*sreg*) jest odpowiedzialny za globalne włączanie i wyłączanie przerwań. Bit ten nazywa się „interrupt bit *i*”.

W diagramie poniżej przedstawiono bity rejestru statusowego.



4.1. Instrukcja „Set interrupts” (sei)

Ustawia flagę „Global Interrupt” (*i*) w *sreg* (rejestr statusowy). Instrukcja następująca po *sei* będzie wykonana przed jakimkolwiek oczekującym przerwaniem.

sei ; set global interrupt enable

4.2. Instrukcja „Clear interrupts” (cli)

Zeruje flagę „Global Interrupt” (*i*) w *sreg* (rejestr statusowy). Przerwania zostaną natychmiast wyłączone. Żadne przerwanie po instrukcji *cli* nie zostanie obsłużone, nawet jeżeli wystąpi jednocześnie z nią. Pojedynczą instrukcją *cli* możemy zabezpieczyć naszą krytyczną część kodu przed przerwaniem.

cli ; disable all interrupts

4.3. Instrukcja „Sleep” (sleep)

Instrukcja ta pozwala na uśpienie procesora.

sleep ; put MCU in sleep mode

Example:

sei ; set global interrupt enable

sleep ; enter sleep, waiting for interrupt

5. Przerwania od Timera

W ćwiczeniu 6 poznaliśmy zasady posługiwania się timerem 0 i timerem 1 za pomocą procedury odpytywania. Przykładowo możemy sprawdzać bit *tof0* poprzez instrukcję *sbrs* aby monitorować przepełnienie timera 0. Podobnie możemy sprawdzać bit *ocf0* aby stwierdzić wystąpienie zgodności z wartością porównywaną dla timera 0 (ang. compare match). Metody te w dużym stopniu ograniczają możliwości obliczeniowe naszego mikrokontrolera. Z wykorzystaniem przerw od timerów możemy znieść te ograniczenia.

Aby włączyć przerwanie od przepełnienia timera, bit **timer overflow interrupt enable (toieN)** w rejestrze **timer interrupt mask (timsk)** musi być ustawiony.

Włączenie przerwania od wystąpienia zgodności z wartością porównywaną dla timera następuje poprzez ustawienie bitu **output compare match interrupt enable (ocieN)** w rejestrze **timsk**.

Example

ldi r20, (1<<toie0) ;enable Timer0 overflow interrupt

out timsk, r20

5.1. Rejestr „Timer interrupt mask” (*timsk*)

Bit	7	6	5	4	3	2	1	0
<i>timsk</i>	<i>ocie2</i>	<i>toie2</i>		<i>ocie1a</i>	<i>ocie1b</i>	<i>toie1</i>	<i>ocie0</i>	<i>toie0</i>

ocie2 D7 Timer/Counter2 Output Compare Match Interrupt Enable

toie2 D6 Timer/Counter2 Overflow Interrupt Enable

ocie1a D4 Timer/Counter1, Output Compare A Match Interrupt Enable

ocie1b D3 Timer/Counter1, Output Compare B Match Interrupt Enable

toie1 D2 Timer/Counter1, Overflow Interrupt Enable

ocie0 D1 Timer/Counter0 Output Compare Match Interrupt Enable

toie0 D0 Timer/Counter0 Overflow Interrupt Enable

Zadanie 7.1

Uwzględniając fakt, że procesor ATmega jest taktowany przez zegar 1MHz oraz wykorzystując Timer1 napisz i uruchom program, który będzie migał diodą LED co sekundę. Wykorzystaj przerwanie od „compare match” w timerze 1.

Podpowiedź: Użyj programu “timer1_1sec.asm” z ćwiczenia 6 jako bazę do budowy nowego rozwiązania.

Gdy skończysz zapisz program pod nazwą “timer1_1sec_int.asm”.

Zadanie 7.2

Napisz program zliczający ilość wciśnień przycisku. W tym zadaniu należy zaproponować rozwiązanie eliminatora drgań ze styków przycisku (ang. debouncer).

Wskazówki.

1. Użyj przerwania od timera 0 do odczytu stanu przycisku co 5ms.
2. W procedurze obsługi przerwania skopiuj stan przycisku do rejestru ogólnego przeznaczenia. Użyj tego rejestru w głównej pętli programu do analizy aktualnego stanu na styku z przyciskiem.
3. Aby brać pod uwagę jedynie pojedyncze wciśnięcia przycisku wykrywaj w programie moment jego wciśnięcia ale poczekaj na jego zwolnienie ze wznowieniem procedury.

6. Przerwania od zdarzeń zewnętrznych

W procesorze ATmega32 występują trzy sprzętowe przerwania od zdarzeń zewnętrznych: Int0, Int1 i Int2. Znajdują się one na wybranych pinach portów. Przerwania te muszą zostać aktywowane zanim będą mogły zostać wykorzystane. Do tego celu wykorzystuje się bit *intN* zlokalizowany w rejestrze *gicr*.

Ext. interrupt	Pin location	Int. vector location	<u>Enable bit location</u>
Int0	Port D.2	\$0002	gicr.6
Int1	Port D.3	\$0004	gicr.7
Int2	Port B.2	\$0006	gicr.5

Example

```
ldi r20, (1<<int0)    ;enable external interrupt 0
out gicr, r20
```

6.1. Rejestr „General Interrupt Control” (gicr)

Przerwania zewnętrzne można włączać niezależnie przy pomocy bitów w rejestrze „General Interrupt Control” (gicr).

Bit	7	6	5	4	3	2	1	0
<i>gicr</i>	<i>int1</i>	<i>Int0</i>	<i>int2</i>					

int1 D7 =0 disabled external interrupt 1

=1 enables external interrupt 1

int0 D6 =0 disabled external interrupt 0

=1 enables external interrupt 0

int2 D5 =0 disabled external interrupt 2

=1 enables external interrupt 2

Bity te wraz z bitem *i* z rejestru *sreg* muszą być ustawione na poziom wysoki aby mikrokontroler zareagował na przerwania.





6.2. Przerwania wyzwalane zboczem a poziomem

Int2 jest jedynym przerwaniem wyzwalanym zboczem, podczas gdy Int0 i Int1 mogą zostać zaprogramowane do wyzwalania zboczem bądź poziomem. W przypadku trybu wyzwalania przerwania poziomem, jeżeli chcemy aby procedura ISR była uruchamiana tylko raz, pin odpowiedzialny za przerwanie musi być dezaktywowany zanim zostanie wykonana instrukcja *reti*.





Zaraz po resece procesora przerwania Int0 i Int1 pracują w trybie wyzwalania poziomem niskim. Jeżeli chcemy zmienić tryb wyzwalania musimy zaprogramować odpowiednie bity „Interrupt Sense Control” (*isc*) w rejestrze *mcucr*.

Bit	7	6	5	4	3	2	1	0
<i>mcucr</i>					<i>isc11</i>	<i>isc10</i>	<i>isc01</i>	<i>isc00</i>

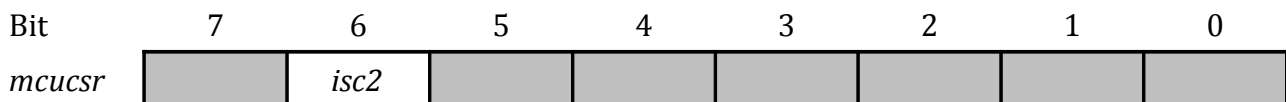
Bity *isc01* i *isc00* są odpowiedzialne za tryb wyzwalania dla przerwania Int0.



isc01	isc00		Description
0	0		Low-level of Int0 pin
0	1		Any logical change on Int0 pin
1	0		Falling edge of Int0 pin
1	1		Rising edge of Int0 pin

Bity *isc11* i *isc10* są odpowiedzialne za tryb wyzwalania dla przerwania Int1.

isc11	isc10		Description
0	0		Low-level of Int1 pin
0	1		Any logical change on Int1 pin
1	0		Falling edge of Int1 pin
1	1		Rising edge of Int1 pin

Bit *isc2* w rejestrze *mcucsr* określa czy przerwanie Int2 będzie aktywowane zboczem narastającym czy opadającym.



isc2		Description
0		Falling edge of Int2 pin
1		Rising edge of Int2 pin

Example

```

;make Int0 rising-edge active
ldi r20, (1<<isc01)|(1<<isc00)
out mcucr, r20

```

6.3. Rejestr „General Interrupt Flag” (*gifr*)

Rejestr *gifr* zawiera flagi wszystkich zewnętrznych zdarzeń dla procesora ATMega. Każdy z tych bitów jest niezależnie ustawiany na poziom wysoki w momencie kiedy nastąpi dane zdarzenie związane z przerwaniami. Należy zwrócić uwagę, że bity te są ustawiane niezależnie

od tego czy zostały aktywowane. Aby anulować wszystkie wcześniejsze zewnętrzne wydarzenia należy wyzerować wszystkie flagi w momencie włączania przerwań.

Bit	7	6	5	4	3	2	1	0
<i>gifr</i>	<i>intf1</i>	<i>intf0</i>	<i>intf2</i>					

Bity *intf0*, *intf1*, i *intf2* są ustawiane w momencie wystąpienia danego przerwania. Flagi te są automatycznie zerowane zaraz po uruchomieniu danej procedury obsługi przerwania. Mogą one zostać również wyzerowane przez nas ręcznie poprzez ustawienie bitów w rejestrze *gifr*.

Example

```
;clear intf1 flag
ldi r20, (1<<intf1)
out gifr, r20
```

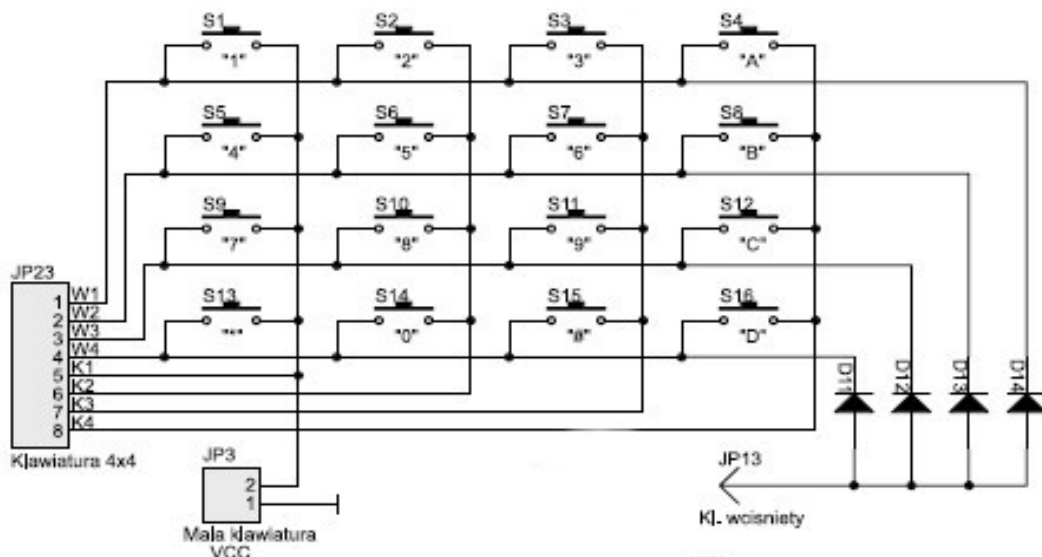
7. Kolejne podejście do klawiatura ZL3AVR

Do tej pory w ćwiczeniach używaliśmy klawiatury ZL2AVR w trybie wersji małej (zwarta zworka JP3). Nadszedł czas aby wykorzystać naszą klawiaturę w pełni. Chcemy być w stanie odczytać wszystkie 16 przycisków. W tym celu musimy stworzyć specjalną procedurę, która uwzględni podłączenie przycisków w charakterze matrycy 4x4.

Aby zdekodować który przycisk został wciśnięty musimy ustalić jednocześnie, która kolumna i który wiersz zostały aktywowane poprzez klawiaturę. Do odczytu **która kolumna jest aktywna** musimy **ustawić linie wierszy (W1..W4) jako wyjścia**, a **linie kolumn (K1..K4) jako wejścia**. Ustawiamy niski poziom na liniach wierszy i odcytujemy linie kolumn. Aktywną kolumnę odcytamy z niskim stanem.

Do odczytu **który wiersz jest aktywny** musimy **ustawić linie kolumn (K1..K4) jako wyjścia**, a **linie wierszy (W1..W4) jako wejścia**. Ustawiamy niski poziom na liniach kolumn i odcytujemy linie wierszy. Aktywny wiersz odcytamy z niskim stanem. Następnie na podstawie odczytanych współrzędnych określamy wciśnięty przycisk.

Dodatkowo klawiatura jest w stanie wygenerować sygnał logiczny w momencie wciśnięcia któregośkolwiek przycisku. Sygnał ten może nam posłużyć jako zewnętrzne przerwanie. Jeżeli linie wierszy (W1..W4) są ustawione jako wyjścia oraz mają stan niski, na JP13 wystąpi niski poziom logiczny w momencie naciśnięcia któregośkolwiek przycisku.



Zadanie 7.3

Napisz program, który wskaże który przycisk z klawiatury został wciśnięty. Wykorzystaj przerwanie od zdarzenia zewnętrznego aby uruchomić procedurę dekodowania. Wyświetl kod wciśniętego przycisku na diodach LED.

Aby uprościć program można założyć, że kod przycisku będzie składał się z kodu wiersza na wyższych 4 bitach, oraz kodu kolumny na 4 niższych bitach. Jako kody wierszy i kolumn można wykorzystać odwrotność kodowania „1 z N” (pozycja 0 determinuje zakodowaną wartość).

7 .. 4	3 .. 0
Column code (one cold)	Row code (one cold)

Row no.	Code	Column no.	Code
1	'0111'	1	'0111'
2	'1011'	2	'1011'
3	'1101'	3	'1101'
4	'1110'	4	'1110'

Podłącz diody LED do portu A, a klawiaturę do portu D.

Port A.0=D0	Port D.0=K4
Port A.1=D1	Port D.1=K3
Port A.2=D2	Port D.2=K2
Port A.3=D3	Port D.3=K1
Port A.4=D4	Port D.4=W4

Port A.5=D5	Port D.5=W3
Port A.6=D6	Port D.6=W2
Port A.7=D7	Port D.7=W1

Użyj zewnętrznego przerwania Int2 do uruchomienia procedury dekodowania klawiatury.

Upewnij się, czy jako źródło zegarowe dla procesora ATmega wybrany jest wewnętrzny oscylator zaprogramowany na częstotliwość 1MHz.

Poniżej przedstawioną propozycję ułożenia kodu programu dla tego zadania.

```
.include "m32def.inc"

.cseg
.org 0

        jmp start

.org INT2addr

        jmp keypad_ISR    ;Keyoad External Interrupt Request 2

.def button_code=r20
; Main program start
start:
    ; Set Stack Pointer to top of RAM
    ???
    ???
    ???
    ???
    ;Set up port A as output for LED controls
    ???
    ???
    ; Clear intf2 flag
    ldi r16, ???
    out gifr, r16
    ; Enable Int2
    ldi r16, ???
    out gicr, r16
    ; Set Int2 active on falling edge
    ldi r16, ???
    out mcucr, r16
    ;Set rows as inputs and columns as outputs
    ???
    ???
    ;Set rows to high (pull ups) and columns to low to activate JP13
```

```

????
????
;Global Enable Interrupt
????
;Set up infinite loop
loop:
    ;read button code from r20 and send to port A
    ???
    rjmp loop

;Keypad Interrupt Service Routine
keypad_ISR:
    ;Disable interrupts
    ???
    ;Store registers that are to be used in ISR on stack
    ???
    ???
    ;Set rows as outputs and columns as inputs on Port D
    ???
    ???
    ;Set rows to low and columns to high (pull up) on Port D
    ???
    ???
    ;Read Port D. Column code in lower nibble
    nop    ;!!!!
    ???
    ;Store column code to r20 on lower nibble
    ???
    ;Set rows as inputs and columns as outputs on Port D
    ???
    ???
    ;Set rows to high (pull up) and columns to low on Port D
    ???
    ???
    ;Read Port D. Row code in higher nibble
    nop ;!!!!
    in r16, ???
    ;Store row code to r20 on higher nibble
    ???
    ???

```

```

????
????
????
;Set rows as inputs and columns as outputs
????
????
;Set rows to high and columns to low to activate JP13
????
????
Clear intf2 flag
ldi r16, ????
out gifr, r16
;Restore registers from stack (in opposite order)
pop ????
pop ????
;Enable interrupts globally
????
reti

```