

Akademia
Górnictwo-Hutnicza
w Krakowie
Instytut Elektroniki



Laboratorium mikrokontrolerów

Ćwiczenie 7

Przerwania

Autor: Paweł Russek
Tłumaczenie: Sebastian Koryciak
ver. 04/21/21

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 przerwania 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.

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 poprzez 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 zrzucony 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 początek programu ISR. Cała przestrzeń w pamięci zawierająca procedury obsługi przerw nazywana jest tablicą wektorów przerw (ang. interrupt vector table).

Poniżej znajduje się fragment tablicy wektorów przerw dla ATmega328PB.

Rodzaj przerwania	lokalizacja w EEPROM	etykieta .equ
Reset	\$0000	
External Interrupt Request 0	\$0002	INT0addr
External Interrupt Request 1	\$0004	INT1addr
Pin Change Interrupt Request 0	\$0006	PCINT0addr
Pin Change Interrupt Request 1	\$0008	PCINT1addr
Pin Change Interrupt Request 2	\$000A	PCINT2addr
Watchdog Time-out Interrupt	\$000C	WDTaddr
Timer/Counter2 Compare Match A	\$000E	TIMER2_COMPAaddr
Timer/Counter2 Compare Match B	\$0010	TIMER2_COMPBaddr
Timer/Counter2 Overflow	\$0012	TIMER2_OVFaddr
Timer/Counter1 Capture Event	\$0014	TIMER1_CAPTaddr
Timer/Counter1 Compare Match A	\$0016	TIMER1_COMPAaddr
Timer/Counter1 Compare Match B	\$0018	TIMER1_COMPBaddr
Timer/Counter1 Overflow	\$001A	TIMER1_OVFaddr
-----	-----	-----
Pin Change Interrupt Request 3	\$0036	PCINT3addr

Kod poniżej przedstawia typowe i najczęściej spotykane ustawienie dla obsługi Resetu i wybranych adresów wektorów przerw dla ATmega328PB. Zawarta w nim jest obsługa następujących przerw: *External Interrupt Request 0*, *Timer/Counter1 Compare Match A*, oraz *Timer/Counter1 Overflow*.

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

Example.

```
.include "m328pbdef.inc"
.org 0
    jmp reset
.org INT0addr                ; External Interrupt Request 0
    jmp external_interrupt_0
.org TIMER1_COMPAaddr        ; Timer/Counter1 Compare Match A
    jmp timer0_compare_match
.org TIMER1_OVFaddr          ; Timer/Counter1 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

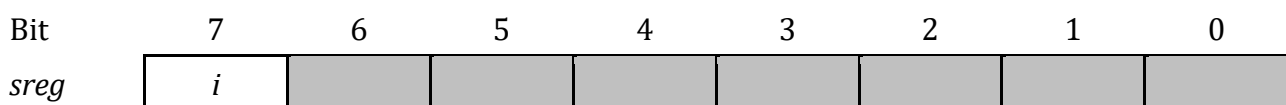
timer0_compare_match
    ;... some code
    reti

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 7 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 zdarzeń zewnętrznych

W procesorze ATmega328PB występuje sześć sprzętowych przerwań od zdarzeń zewnętrznych. Dwa z nich (Int0 i Int1) przypisane są do pojedynczych pinów, natomiast pozostałe cztery (PCINT0, PCINT1, PCINT2 i PCINT3) do wszystkich dostępnych portów mikrokontrolera. Przerwania te muszą zostać aktywowane i skonfigurowane zanim będą mogły zostać wykorzystane. Do tego celu wykorzystuje się serię rejestrów, które zostały zgromadzone w tabeli poniżej i zostaną opisane w dalszej części tej instrukcji.

Ext. interrupt	Pin/port location	Int. vector location	Registers	
INT0	Port D pin 2	\$0002	EIMSK, EICRA, EIFR	
INT1	Port D pin 3	\$0004		
PCINT0	Port B	\$0006	PCICR, PCIFR	PCMSK0
PCINT1	Port C	\$0008		PCMSK1
PCINT2	Port D	\$000A		PCMSK2
PCINT3	Port E	\$0036		PCMSK3

Example

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

5.1. Rejestr „External Interrupt Mask” (EIMSK)

Przerwania zewnętrzne można włączać niezależnie przy pomocy bitów w rejestrze „External Interrupt Mask” (EIMSK).

Bit	7	6	5	4	3	2	1	0
<i>eimsk</i>							<i>int1</i>	<i>int0</i>

int1 b1 = 0 disabled external interrupt 1
= 1 enables external interrupt 1

int0 b0 = 0 disabled external interrupt 0
= 1 enables external interrupt 0

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





5.2. Przerwania wyzwalane zboczem lub poziomem (INT0 i INT1)

Przerwania *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 resecie 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 *EICRA*.

Bit	7	6	5	4	3	2	1	0
<i>eicra</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*.

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

Bity *isc11* i *isc10* są odpowiedzialne za tryb wyzwalania dla przerwania *Int1*, a ich działanie i sposób ustawiania jest taki sam jak dla przerwania *Int0*.

Dostęp do rejestru *eicra* następuje z wykorzystaniem instrukcji *sts/lds* (ponieważ adres rejestru = 0x69 > 0x3F).

Example

```
;make Int0 rising-edge active
ldi r20, (1<<isc01)|(1<<isc00)
sts eicra, r20
```

5.3. Rejestr „External Interrupt Flag” (EIFR)

Rejestr *EIFR* zawiera flagi zewnętrznych zdarzeń *Int0* i *Int1* dla procesora ATmega. Każdy z tych bitów jest niezależnie ustawiany na poziom wysoki w momencie, kiedy nastąpi dane zdarzenie związane z przerwaniem. 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>eifr</i>							<i>intf1</i>	<i>intf0</i>

Bity *intf0*, *intf1* 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 (wpisywanie '1') bitów w rejestrze *eifr*.

Example

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

5.4. Rejestr „Pin Change Interrupt Control” (PCICR)

Przerwania zewnętrzne typu „pin change” można włączać niezależnie dla każdego z portów przy pomocy bitów w rejestrze „Pin Change Interrupt Control” (*PCICR*).

Bit	7	6	5	4	3	2	1	0
<i>pcicr</i>					<i>pcie3</i>	<i>pcie2</i>	<i>pcie1</i>	<i>pcie0</i>

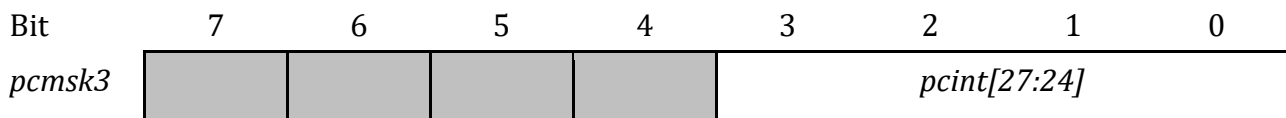
Bity te wraz z bitem *I* z rejestru *sreg* muszą być ustawione na poziom wysoki, aby mikrokontroler zareagował na przerwanie. Dodatkowo w obrębie każdego z portów należy wybrać, które z pinów będą aktywne – służą do tego rejestry *pcmskx*.

Rejestr *pcicr* ma adres 0x68 dlatego do jego odczytu/zapisu należy użyć instrukcji *sts/lds*.

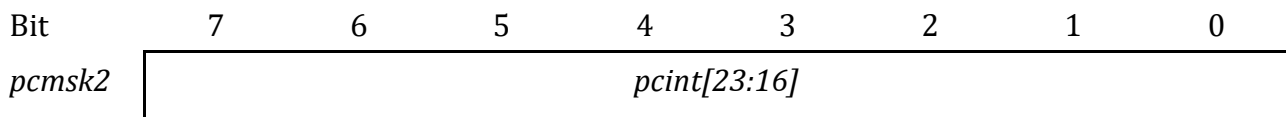
5.5. Rejestry „Pin Change Mask” (PCMSKx)

Przerwania zewnętrzne typu „pin change” podzielone są na porty: E - PCINT[27:24], D - PCINT[23:16], C - PCINT[14:8], B - PCINT[7:0]. Mają one różną długość, ponieważ porty w mikrokontrolerze posiadają określoną ilość pinów. Wadą przerwań tego typu jest mała możliwość ich konfiguracji (reakcja tylko na zmianę stanu). Ich zaletą jest to, że podłączone są do nich wszystkie piny, którymi dysponuje mikrokontroler. Każdemu portowi przypisana jest jedna procedura obsługi przerwania. W obrębie każdego z portów możemy zdecydować, które piny będą brały udział w aktywacji przerwania – służą do tego rejestry *pcmskx*.

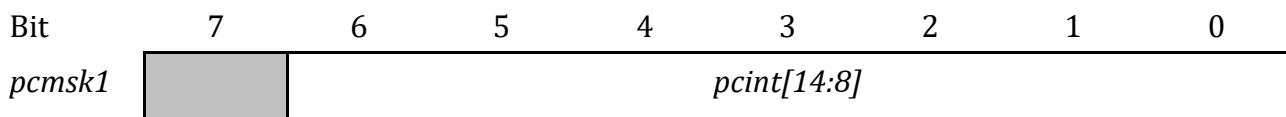
PORT E[3:0]



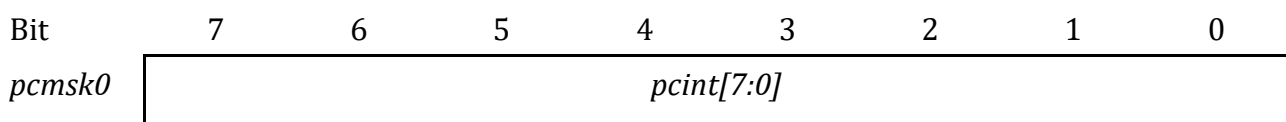
PORT D[7:0]



PORT C[6:0]



PORT B[7:0]

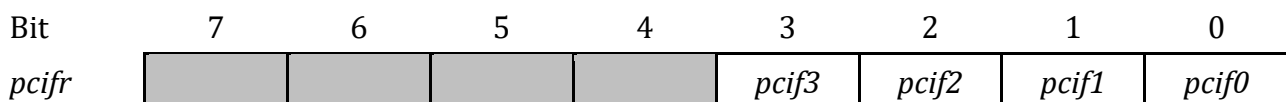


Example

```
;make interrupt on porte.2 and porte.0 active  
ldi r20, (1<<pcint26)|(1<<pcint24)  
sts pcmsk3, r20 ;set masks on pcint[27:24] = porte  
ldi r20, (1<<pcie3)  
sts pcicr, r20 ;enable interrupt
```

5.6. Rejestr „Pin Change Interrupt Flag” (PCIFR)

Rejestr *PCIFR* zawiera flagi zewnętrznych zdarzeń *PCINT0*, *PCINT1*, *PCINT2* i *PCINT3* dla procesora ATmega. Każdy z tych bitów jest niezależnie ustawiany na poziom wysoki w momencie, kiedy nastąpi dane zdarzenie związane z przerwaniem. 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.

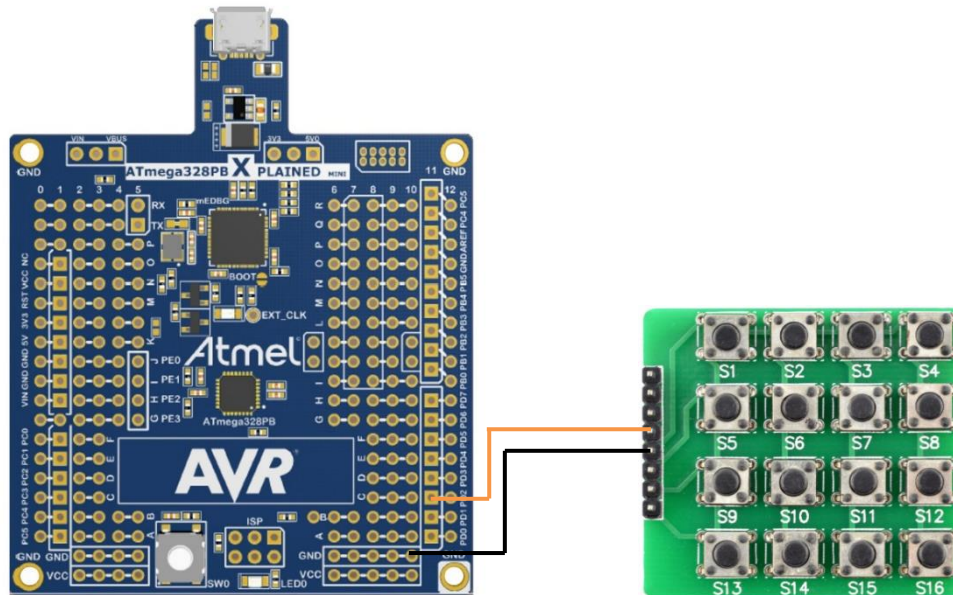


Bity te są ustawiane w momencie wystąpienia danego przerwania. Flagi są automatycznie zerowane zaraz po uruchomieniu danej procedury obsługi przerwania. Mogą one zostać również wyzerowane przez nas ręcznie poprzez ustawienie (wpisywanie '1') bitów w rejestrze *pcifr*.

6. Sterowanie diodą LED z przerwaniami

Do zadania wykorzystana zostanie dioda LED znajdująca się na platformie Xplained – LED0. Jest ona podłączona do pinu 5 portu B.

Proszę podłączyć przycisk S0 z zewnętrznej klawiatury do platformy Xplained zgodnie z rysunkiem (pin 2 portu D, do którego podłączone jest INT0).



Zadanie 7.1

Napisz program, który zmieni stan diody LED0 na przeciwny w odpowiedzi na przerwanie zewnętrzne wywołane przez naciśnięcie przycisku S0. Możesz do tego celu użyć przykładowy kod umieszczony poniżej.

```
#include "m328pbddef.inc"
.org 0
    jmp main                ;skip vector table
.org ??????
    jmp int0_isr
;----- main -----
main:
    ldi r16, LOW(RAMEND)    ;initialize stack for ISR
    out spl, r16
    ldi r16, HIGH(RAMEND)
    out sph, r16
    sbi ddrb, 5             ;portb.5 is output (led0)
    sbi portd, 2            ;pull-up enable for portd.2
    ldi r20, (1<<?????)
    out eimsk, r20          ;enable int0
    ldi r20, (1<<?????)
    sts eicra, r20          ;set int0 active on falling edge
    sei                    ;enable interrupts
stop:
    jmp stop                ;stay forever
;----- int0 ISR -----
int0_isr:
    in r21, pinb            ;read portb
```

```
ldi r22, 0x20
eor r21, r22           ;toggle bit 5
out portb,r21
reti
```

Zadanie 7.2

Zmodyfikuj program tak, aby dioda zmieniała swój stan w odpowiedzi na przycisk SW0 umieszczony na platformie Xplained. Przycisk ten jest podłączony do pinu 7 portu B.

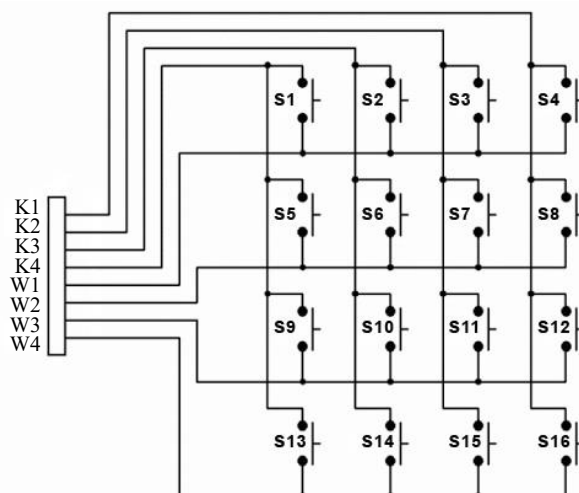
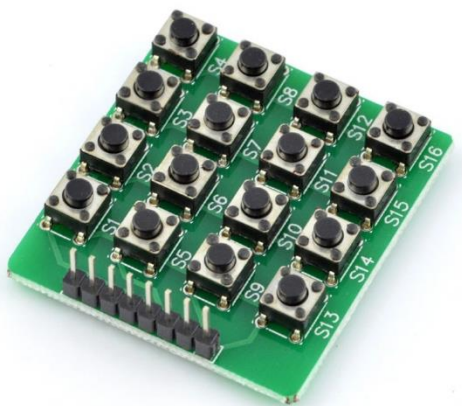
Jaka jest różnica w działaniu programu? Jak to naprawić?

7. Kolejne podejście do modułu klawiatury

Do tej pory w ćwiczeniach używaliśmy maksymalnie 4 przycisków z zewnętrznej klawiatury (przyciski S1-S4). Nadszedł czas, aby wykorzystać naszą klawiaturę w większym zakresie. Chcemy być w stanie odczytać 8 przycisków ułożonych w 4 kolumny. W tym celu musimy stworzyć specjalną procedurę, która uwzględni podłączenie przycisków w charakterze matrycy 4x2.

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, W2) jako wyjścia**, a **linie kolumn (K1...K4) jako wejścia**. Ustawiamy niski poziom na liniach wierszy i odczytujemy linie kolumn. Aktywną kolumnę odczytamy z niskim stanem.

Do odczytu **który wiersz jest aktywny** musimy **ustawić linie kolumn (K1...K4) jako wyjścia**, a **linie wierszy (W1, W2) jako wejścia**. Ustawiamy niski poziom na liniach kolumn i odczytujemy linie wierszy. Aktywny wiersz odczytamy z niskim stanem. Następnie na podstawie odczytanych współrzędnych określamy wciśnięty przycisk.



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 2 bitach, oraz kodu kolumny na 4 niższych bitach. Jako kody wierszy i kolumn można wykorzystać kodowanie „1 z N” (pozycja 1 determinuje zakodowaną wartość).

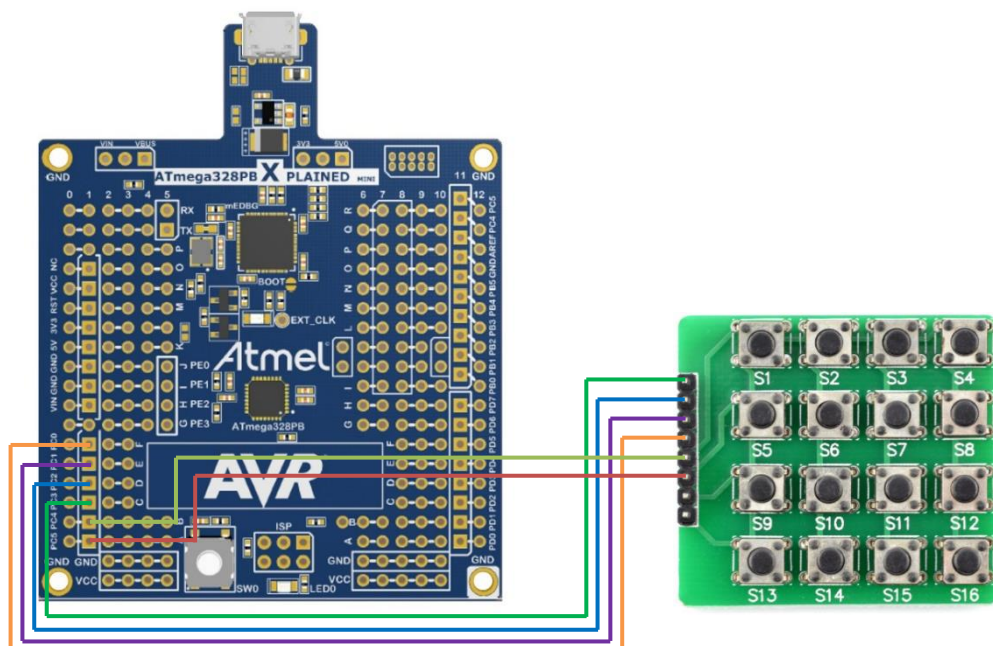
5, 4	3, 2, 1, 0
Row code (one hot)	Column code (one hot)

Row no.	Code
1	'01'
2	'10'

Column no.	Code
1	'0001'
2	'0010'
3	'0100'
4	'1000'

Podłącz diody LED do portu B, a klawiaturę do portu C.

Port B.0=D0	Port C.0=K4
Port B.1=D1	Port C.1=K3
Port B.2=D2	Port C.2=K2
Port B.3=D3	Port C.3=K1
Port B.4=D4	Port C.4=W1
Port B.5=D5	Port C.5=W2



Użyj zewnętrznego przerwania typu „Pin Change” do uruchomienia procedury dekodowania klawiatury.

Poniżej przedstawiono propozycję ułożenia kodu programu dla tego zadania.

```
.include "m328pbdef.inc"

.cseg
.org 0
    jmp start
.org 0x0000
    rjmp keypad_ISR ;Keypad External Interrupt Request

;-----
;Initialization
start:

    ; Set Stack Pointer to top of RAM
    ldi r16, high(0x0000)
    out SPH, r16
    ldi r16, low(0x0000)
    out SPL, r16

    ;SET UP 6 LEDS
    ;Set up port B as output for LED controls
    ldi r16, 0x3F
    out ddrb, r16

    ;SET UP KEYPAD, 2 rows x 4 cols
    ;Set rows as inputs and columns as outputs
    ldi r20, 0x0f
    out ddrc, r20

    ;Set rows to high (pull ups) and columns to low
    ldi r20, 0x30
    out portc, r20

    ;Select rows as interrupt triggers
    ldi r20, (1<<PCINT0)|(1<<PCINT1)
    sts PCMSK1, r20

    ;Enable PCINT1
    ldi r20, (1<<PCIE1)
    sts PCICR, r20

    ;Reset register for output
    ldi r18, 0x00

    ;Global Enable Interrupt
    Sei

;-----
;Set up infinite loop
loop:

    call led_display
    rjmp loop

;-----
;Keypad Interrupt Service Routine
keypad_ISR:

    ;Set rows as outputs and columns as inputs
    ldi r20, 0x000f
    out ddrc, r20
```

```

;Set columns to high (pull ups) and rows to low
ldi r20, 0x????
out portc, r20

;Read Port C. Columns code in low nibble
in r16, pinc

;Store columns code to r18 on low nibble
mov r18, r16
andi r18, 0x0f

;Set rows as inputs and columns as outputs
ldi r20, 0x00ff
out ddrc, r20

;Set rows to high (pull ups) and columns to low
ldi r20, 0xffff
out portc, r20

;Read Port C. Rows code in high nibble
in r16, pinc

;Merge with previous read
andi r16, 0x30
add r18, r16

reti
;-----
;display value from r18 on leds
led_display:
    out portb, r18
    ret

```

Zadanie 7.4*

Napisz program, który wskaże który przycisk z klawiatury został wciśnięty. Tym razem podłącz całą klawiaturę 4x4. Wykorzystaj przerwanie od zdarzenia zewnętrznego, aby uruchomić procedurę dekodowania. Wyświetl kod wciśniętego przycisku tym razem na wyświetlaczu 7 segmentowym w postaci heksadecymalnej (0-F).