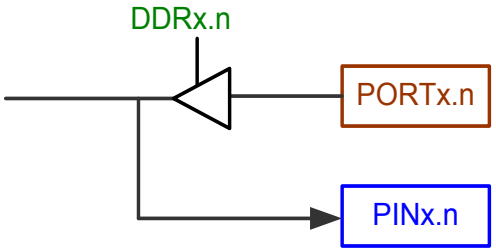
1. Zagadnienia

- Linie I/O jako wejścia
- Stan wysokiej impedancji, rezystory *pull-up*
- Obsługa przycisków i klawiatury matrycowej
- Symulacje w programie SimulIDE

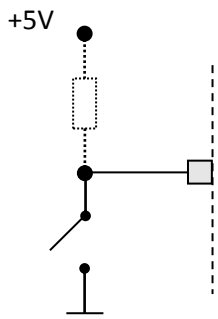
2. Linie I/O

Aby linie wybranego portu pracowały jako wejścia należy w rejestrze kierunkowym DDRx ustawić "0" na pozycji odpowiadającej danej linii. Kierunek każdej linii można ustalić indywidualnie, tak więc w jednym porcie część linii może pracować jako wejścia, część jako wyjścia. Jeżeli linie ustawione są jako wejściowe, odczyt stanu linii dokonujemy z wykorzystaniem rejestru PINx.

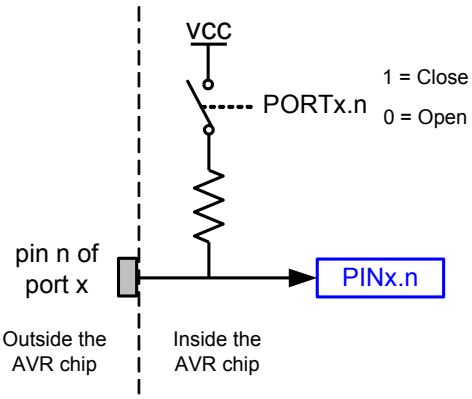


DDRx - rejestr kierunkowy
PORTx - rejestr wyjściowy
PINx - rejestr wejściowy

3. Rezystory *pull-up*



Aby dołączyć prosty przycisk do linii wejściowej, zazwyczaj podłączamy go drugim wyprowadzeniem do masy (GND). Naciśnięcie przycisku powoduje wymuszenie "0" (stanu niskiego) na wejściu. Natomiast, jaką wartość odczyta mikrokontroler, gdy przycisk nie jest naciśnięty (wejście będzie "wisiało w powietrzu")? Będzie to stan nieznany, niejednoznaczny z punktu widzenia logiki zero-jedynkowej, określany jako stan wysokiej impedancji (*tri-state*, *Hi-Z*). Aby uniknąć takich niejednoznacznych sytuacji na wejściu stosuje się rezystory podciągające (*pull-up*), które dołączone do +5V ("1" logicznej) wymuszają stan wysoki, gdy przycisk jest rozwarty.



W mikrokontrolerze ATmega16 (zamiast stosowania rezystorów zewnętrznych) mamy możliwość programowego załączenia/odłączenia wewnętrznego rezystora *pull-up* przypisanego do każdej linii. Możemy zatem wybrać, czy linia wejściowa będzie w stanie wysokiej impedancji, czy w stanie "1" (możliwość ustawienia linii w stan wysokiej impedancji wykorzystuje się, np. przy dołączaniu linii portu do magistrali). W przypadku pracy z przyciskami (klawiaturą) konieczne jest zastosowanie zewnętrznych lub wewnętrznych, rezystorów podciągających *pull-up*. Różne sposoby programowego dołączania/odłączania wewnętrznych rezystorów *pull-up*, opisane są w nocie katalogowej ATmega16 (*Tabela 20, str. 52*).

4. Obsługa przycisków

Napisz program prezentujący stan przycisków dołączonych do portu A, na diodach LED dołączonych do portu B.

Program_01

```
// ----------------------------
// test portów in-out
//-----------------------------

.nolist
.include "m16def.inc"
.list
.listmac
.device ATmega16

.cseg
.org 0x0000
jmp START

;wektory przerwań

.org 0x0030
START:
;ustawienie stosu
ldi r16, high(RAMEND)       ;0x04
out SPH, r16
ldi r16, low(RAMEND)        ;0x5f
out SPL, r16

; ustawienie portu B jako wyjścia
ldi r16, 0b11111111
out DDRB, r16

; ustawienie portu A jako wejścia
ldi r16, 0b00000000
out DDRA, r16


MAIN:

in r16, PINA
out PORTB, r16

jmp MAIN
```

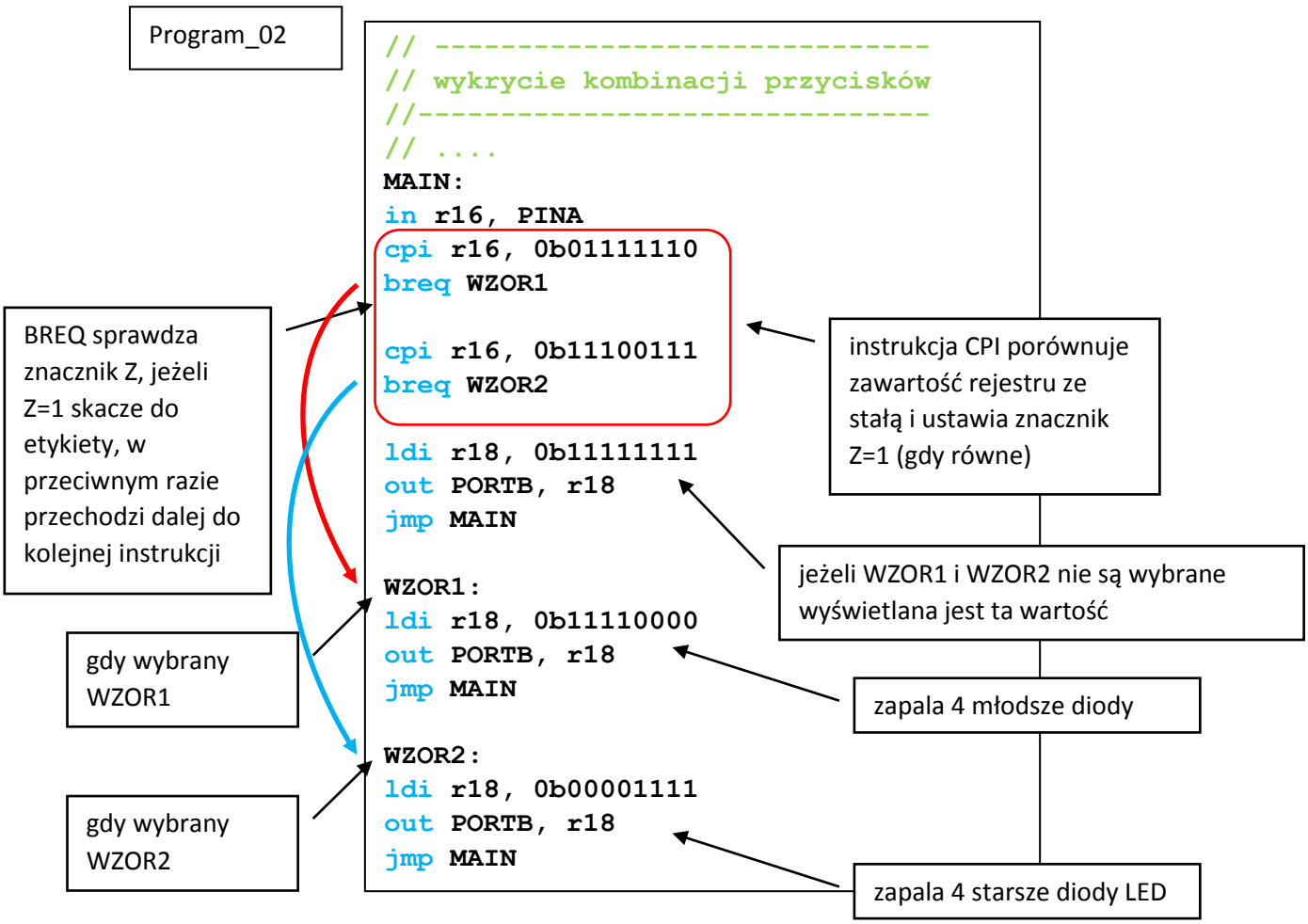
ustawienie portów

odczyt stanu portu A i wysłanie na port B

Program jest na tyle prosty, że nie wymaga specjalnego wyjaśnienia. Należy pamiętać, że do odczytu stanu wejść portu wykorzystujemy rejestr PINx, do wysłania danych na port - rejestr PORTx.

5. Detekcja stanu przycisków

Wykrycie, który przycisk (jeden lub więcej) jest aktualnie wciśnięty, wymaga porównania odczytanego stanu wejść rejestru PINx z oczekiwanym wzorcem i podjęcia decyzji o dalszym działaniu programu (zastosowanie instrukcji warunkowej). Poniższy program (fragment - program główny) testuje stan linii portu A i porównuje ze wzorcem: zapala 4 młodsze diody, gdy wykryto naciśnięcie przycisków 1 i 7; zapala 4 starsze diody, gdy wykryto naciśnięcie przycisków 4 i 5.



W programie wykorzystano instrukcję CPI (porównania bezpośrednio ze stałą), oraz instrukcji skoku warunkowego BREQ (testowanie znacznika Z). Nie jest to oczywiście jedyne rozwiązanie, w instrukcjach asemblera znajduje się wiele różnych instrukcji porównujących i warunkowych (testujących różne znaczniki), które można wykorzystać w programie testowania stanu przycisków (klawiatury). Podane kombinacje przycisków (1,7) i (4,5) oraz wyświetlone wzory są przykładowe, można przetestować własne ustawienia.

Kolejny program (fragment) testuje linie portu A i zapala taką liczbę diod LED, która odpowiada numerowi wybranego przycisku. W tym przypadku wykorzystano instrukcje warunkowe SBIS/SBIC bezpośredniego testowania pojedynczej linii portu I/O.

Program_03

```
// ----------------------------
// testowanie stanu linii portu
//-----------------------------
// ....

MAIN:
ldi r18, 0b11111111

sbis PINA, 0
ldi r18, 0b11111110
sbis PINA, 1
ldi r18, 0b11111100
sbis PINA, 2
ldi r18, 0b11111000
sbis PINA, 3
ldi r18, 0b11110000
sbis PINA, 4
ldi r18, 0b11100000
sbis PINA, 5
ldi r18, 0b11000000
sbis PINA, 6
ldi r18, 0b10000000
sbis PINA, 7
ldi r18, 0b00000000

out PORTB, r18

jmp MAIN
```

wstępnie do r18 wpisany wzór wygaszający wszystkie diody

testowany bit 0 w rejestrze PINA

testowany bit 1 w rejestrze PINA

wzór do wyświetlenia

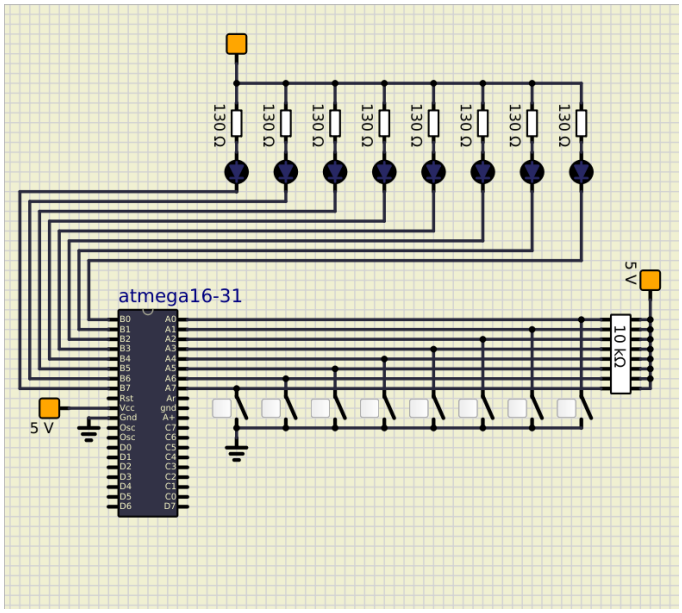
"zapalenie" diod LED kombinacją zapisaną w r18

instrukcja SBIS testuje wybrany bit. Jeżeli warunek jest spełniony (wykryto "1") pomija kolejną instrukcję, jeżeli warunek nie jest spełniony (wykryto "0") nie pomija kolejnej instrukcji pomija kolejną instrukcję gdy

Na wstępie wpisano (LDI r18,0b11111111**),**domyślny wzór do wyświetlenia w przypadku, gdy żaden przycisk nie zostanie przyciśnięty (akurat jest to wygaszenie wszystkich diod LED, ale można wybrać dowolny inny wzór). Instrukcje SBIS (SBIC) testują bezpośrednio wskazany bit rejestru wejściowego PINx (czyli stan linii I/O), i w zależności od spełnienia/niespełnienia warunku pomijają kolejną instrukcję. Nie ma tu możliwości skoku do wskazanej etykiety, a jedynie pominięcie lub nie, kolejnej (jednej) instrukcji - należy to uwzględnić przy tworzeniu algorytmu programu. Jeżeli przycisk nie został naciśnięty, instrukcja jest pomijana, jeżeli przycisk został naciśniety - instrukcja nie jest pomijana, odpowiednia sekwencja jest wpisywana do r18. Po przetestowaniu wszystkich ośmiu przycisków, sekwencja z r18 przesyłana jest do portu B (diody LED). W przypadku gdy żaden przycisk nie został wybrany, w r18 mamy domyślną, wpisaną na początku sekwencję.

6. Symulacja działania w programie SimulIDE

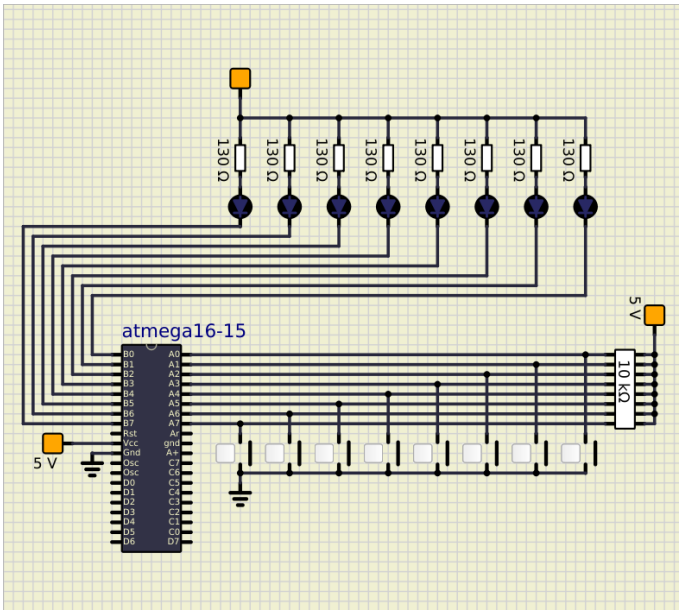
Narysuj schemat zawierający mikroprocesor ATmega16 z podłączonymi diodami LED do portu B. Dobierz odpowiednią wartość rezystorów obliczoną w zależności od koloru wyświetlacza (**Prawy-Przycisk-Myszy > Properties > Resistance**). Do portu A dołącz przyciski (możesz wybrać przyciski niestabilne lub stabilne - dla chętnych również możliwość zastosowania jednorzędowej, 8-pozycyjnej klawiatury).



Podczas symulacji w programie SimulIDE można zastosować przyciski niestabilne lub stabilne (w przypadku zastosowania przycisków stabilnych mamy możliwość symulowania naciśnięcia kilku przycisków jednocześnie). Na schematach zastosowano również rezystory podciągające 10kΩ. (symulator SimulIDE nie daje możliwości symulacji wykorzystania wewnętrznych rezystorów *pull-up*).

"Wgraj" do procesora plik *.hex (plik wynikowy wygenerowany podczas pracy w środowisku AVR Studio). Na schemacie wybierz element procesora, (**Prawy-Przycisk-Myszy > Load firmware**). Ustaw odpowiednią częstotliwość pracy procesora (**Prawy-Przycisk-Myszy > Properties > MHz).**

Uruchom symulację w czasie rzeczywistym, sprawdź poprawność działania programów.



7. Obsługa klawiatury matrycowej 4x4

Prezentowany program obsługuje klawiaturę 16 pozycyjną (4x4) dołączoną do portu A, numer naciśniętego przycisku prezentowany jest na wyświetlaczu 7-segmentowym dołączonym do portu B.

Program_04

```
// ----------------------------
// klawiatura 4x4
// ----------------------------

.nolist          ; komentarz
.include "m16def.inc"
.list
.listmac
.device ATmega16

// definicja stałych
.equ c_0 = 0b11000000
.equ c_1 = 0b11111001
.equ c_2 = 0b10100100
.equ c_3 = 0b10110000
.equ c_4 = 0b10011001
.equ c_5 = 0b10010010
.equ c_6 = 0b10000010
.equ c_7 = 0b11111000
.equ c_8 = 0b10000000
.equ c_9 = 0b10010000
.equ c_A = 0b10001000
.equ c_B = 0b10000011
.equ c_C = 0b11000110
.equ c_D = 0b10100001
.equ c_E = 0b10000110
.equ c_F = 0b10001110

.cseg
.org 0x0000
jmp 0x0030

;wektory przerwań

.org 0x0030
;ustawienie stosu
ldi r16, high(RAMEND) ;0x04
out SPH, r16
ldi r16, low(RAMEND)  ;0x5f
out SPL, r16

; ustawienie portu A 7-4
; wejścia, 3-0 wyjścia
ldi r16, 0b00001111
out DDRA, r16

; ustawienie portu B
ldi r16, 0b11111111
out DDRB, r16
out PORTB, r16   ;wygaszenie
;wszystkich segmentów

MAIN:
//WIERSZ 1
ldi r16, 0b11111110
out PORTA, r16
call JAKI_PRZYCISK

//WIERSZ 2
ldi r16, 0b11111101
out PORTA, r16
call JAKI_PRZYCISK

//WIERSZ 3
ldi r16, 0b11111011
out PORTA, r16
call JAKI_PRZYCISK

// WIERSZ 4
ldi r16, 0b11110111
out PORTA, r16
call JAKI_PRZYCISK

jmp MAIN

.org 0x0100
JAKI_PRZYCISK:
in r17, PINA
;spr. przycisków wiersza 1
cpi r17, 0b11101110
breq KEY_0            ;TAK
                      ;NIE
cpi r17, 0b11011110
breq KEY_1            ;TAK
                      ;NIE
cpi r17, 0b10111110
breq KEY_2            ;TAK
                      ;NIE
cpi r17, 0b01111110
breq KEY_3            ;TAK
                      ;NIE
```

ustawienie portów

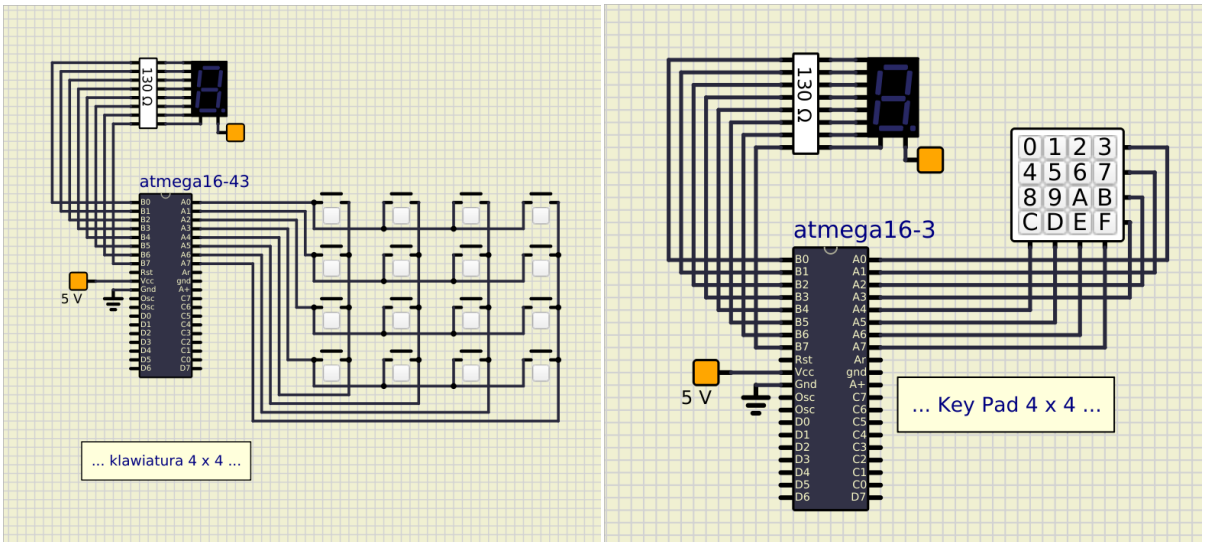
```
;spr. przycisków wiersza 2
cpi r17, 0b11101101
breq KEY_4              ;TAK
                        ;NIE
cpi r17, 0b11011101
breq KEY_5              ;TAK
                        ;NIE
cpi r17, 0b10111101
breq KEY_6              ;TAK
                        ;NIE
cpi r17, 0b01111101
breq KEY_7              ;TAK
                        ;NIE
;spr. przycisków wiersza 3
cpi r17, 0b11101011
breq KEY_8              ;TAK
                        ;NIE
cpi r17, 0b11011011
breq KEY_9              ;TAK
                        ;NIE
cpi r17, 0b10111011
breq KEY_A              ;TAK
                        ;NIE
cpi r17, 0b01111011
breq KEY_B              ;TAK
                        ;NIE
;spr. przycisków wiersza 4
cpi r17, 0b11100111
breq KEY_C              ;TAK
                        ;NIE
cpi r17, 0b11010111
breq KEY_D              ;TAK
                        ;NIE
cpi r17, 0b10110111
breq KEY_E              ;TAK
                        ;NIE
cpi r17, 0b01110111
breq KEY_F              ;TAK
                        ;NIE

ldi r16, 0b11111111
out PORTB, r16  ;wygaszenie
wszystkich segmentów

ret
```

```
KEY_0:
ldi r16, c_0
out PORTB, r16
ret

KEY_1:
ldi r16, c_1
out PORTB, r16
ret

KEY_2:
ldi r16, c_2
out PORTB, r16
ret

KEY_3:
ldi r16, c_3
out PORTB, r16
ret

KEY_4:
ldi r16, c_4
out PORTB, r16
ret

KEY_5:
ldi r16, c_5
out PORTB, r16
ret

KEY_6:
ldi r16, c_6
out PORTB, r16
ret

KEY_7:
ldi r16, c_7
out PORTB, r16
ret

KEY_8:
ldi r16, c_8
out PORTB, r16
ret

KEY_9:
ldi r16, c_9
out PORTB, r16
ret
```

```
KEY_A:
ldi r16, c_A
out PORTB, r16
ret

KEY_B:
ldi r16, c_B
out PORTB, r16
ret

KEY_C:
ldi r16, c_C
out PORTB, r16
ret

KEY_D:
ldi r16, c_D
out PORTB, r16
ret

KEY_E:
ldi r16, c_E
out PORTB, r16
ret

KEY_F:
ldi r16, c_F
out PORTB, r16
ret
```

Mimo, że początkowo program może wydawać się skomplikowany - w rzeczywistości jest dość prosty, składa się z powielonych fragmentów testowania wszystkich 16 przycisków klawiatury.

W programie głównym wysyłamy kolejno (sekwencyjnie) "0" na cztery linie PA0..3 dołączone do wierszy klawiatury. Za każdym razem po "wybraniu" wiersza sprawdzamy czy nie został naciśnięty któryś z klawiszy przypisany do danego wiersza. W tym celu wywoływany jest podprogram JAKI_PRZYCISK w którym odczytujemy aktualny stan portu A - odczytaną sekwencję porównujemy ze wzorcem przypisanym do każdego przycisku. Naciśnięcie przycisku powoduje, że "0" pojawi się na linii przypisanej do kolumny danego przycisku. Zatem każdy przycisk identyfikowany jest na podstawie stanu 8-bitów: 4 wysyłanych wskazujących wybrany wiersz oraz 4 odebranych identyfikujących kolumnę. Jeżeli odczytana sekwencja odpowiada wzorcowi przypisanemu do przycisku, przechodzimy do fragmentu (etykiety) obsługującej dany przycisk. Jeżeli żaden przycisk nie został zidentyfikowany wyświetlana jest domyślna sekwencja. Ponieważ przejście do etykiety wywoływane jest w podprogramie, każdy fragment zakończony jest instrukcją RET - wychodzimy z podprogramu i wracamy w programie głównym do testowania kolejnego wiersza.



Symulacje w programie SimulIDE można wykonać z wykorzystaniem klawiatury zbudowanej z przycisków, pokazującą ideę konstrukcji klawiatury matrycowej i podłączenia sygnałów sterujących do wierszy i kolumn. Można też wykorzystać gotowy element Key Pad dostosowując jego parametry (liczba wierszy i kolumn, opis przycisków).

8. Zadania do samodzielnej realizacji

1. Programy

- **Zadanie 1**: Napisz program prezentujący na wyświetlaczu 7-segmentowym dołączonym do portu B, numer aktualnie naciśniętego przycisku (jednego z ośmiu dołączonych do portu A). Możesz pokazać cyfry 0..7 lub 1..8. Zasymuluj działanie w programie SimulIDE. (Wskazówka: możesz wykorzystać instrukcje SBIC/SBIS do testowania każdej linii osobno i podjąć decyzję o wyświetleniu odpowiedniej cyfry, lub możesz odczytać cały rejestr PINA, porównać ze wzorcem i podjąć decyzję. Obsługa wyświetlacza 7-segmentowego - skorzystaj z rozwiązań z Ćwiczenia 8).
- **Zadanie 2**: Zmodyfikuj program "Biegająca dioda" (Ćwiczenie 7) tak, aby kierunek ruchu diody lewo/prawo był uzależniony od stanu "1" lub "0", jednego przycisku dołączonego do linii PA0 portu A. Zasymuluj działanie w programie SimulIDE.
- **Zadanie 3**: Zadanie podane przez prowadzącego.

2. Zadania dodatkowe (dla chętnych)

- **Zadanie A**: Napisz program obsługi klawiatury windy w wieżowcu 16 kondygnacyjnym, prezentujący na dwóch wyświetlaczach 7-segmentowych (2 cyfry) dziesiętnie numer naciśniętego przycisku klawiatury 4x4 (0..15 lub 1..16). Wyświetlacze dziesiątek i jedności możesz podłączyć do dwóch oddzielnych portów, lub w wersji bardziej zaawansowanej - zastosować 2-cyfrowy wyświetlacz multipleksowany. Wskazówka: program będzie niewielką modyfikacją przykładu Program_04. Zasymuluj działanie w programie SimulIDE.
- **Zadanie B**: Zadanie podane przez prowadzącego.

9. Sprawozdanie

- W sprawozdaniu należy umieścić proste algorytmy oraz kody programów z zajęć i zadań z odpowiednim wyjaśnieniem działania zastosowanych dyrektyw i instrukcji.