Zestaw Edukacyjny Atmega-8
(AJAWe-0711)

LEKCJA 4

Porty wejścia-wyjścia.

W poprzedniej lekcji napisaliśmy pierwszy program, który zapalił nam jedną diodę led. Teraz omówimy szczegółowo działanie niniejszego programu. Nasz program wyglądał następująco:

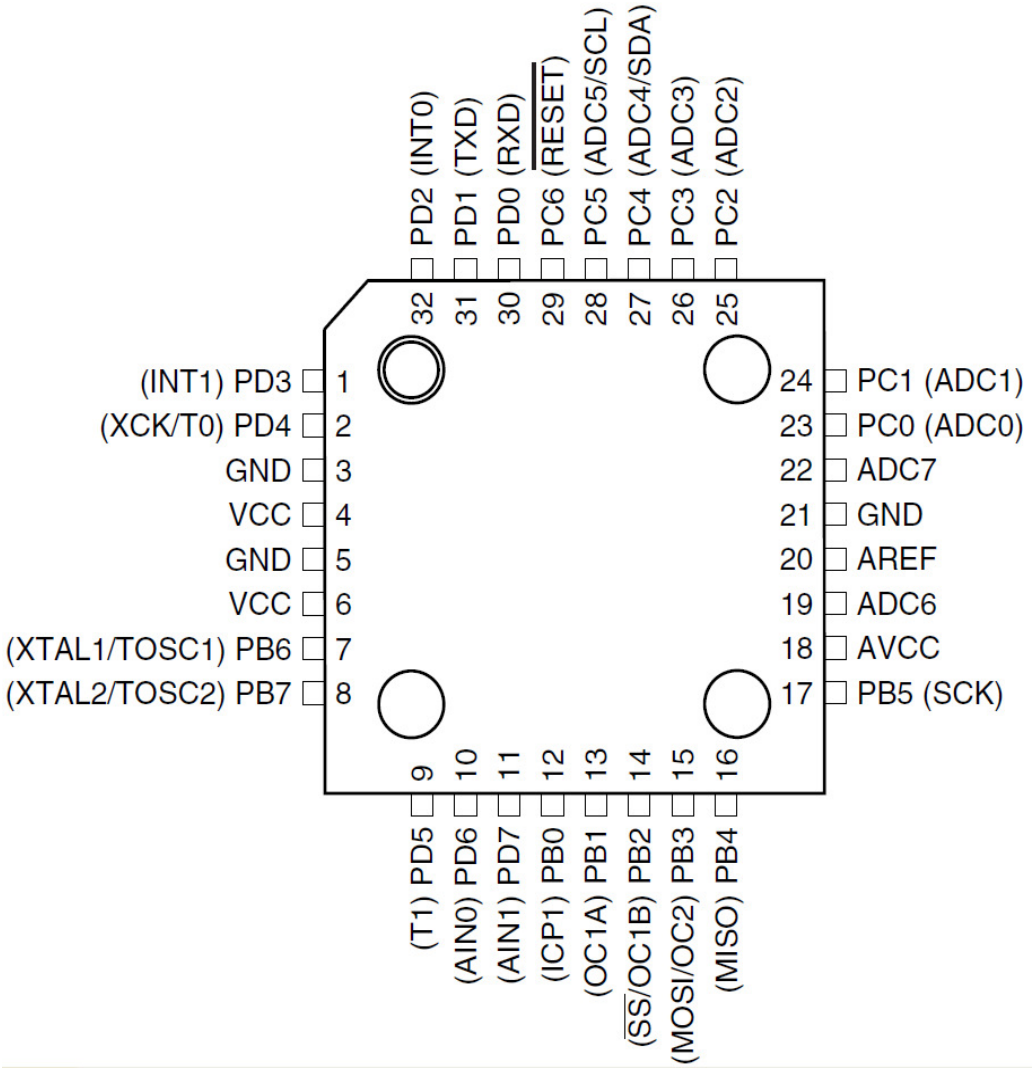
```
//Mój pierwszy program
//Zapal diodę D7 podłączoną do pinu PC2

DDRC  = 0b00000100;
PORTC = 0b11111011;
```

Dwie pierwsze linijki to komentarz, dwie kolejne to właściwy program.
Do każdego portu mikrokontrolera przypisane są po trzy rejestry bajtowe, czyli komórki pamięci zawierające liczbę od 0 do 255. Rejestr **DDRx** to rejestr sterujący, **PORTx** to rejestr stanu wyjścia, **PINx** to rejestr stanu wejścia. Mikrokontroler ATMEGA-8 ma porty oznaczone literkami B,C i D.

Dla portu **B** są to piny PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7.
Dla portu **C**: PC0, PC1, PC2, PC3, PC4, PC5, PC6.
Dla portu **D**: PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7.

Oznaczenia zawarte w nawiasach to alternatywne funkcje jakie mogą być realizowane przez konkretne wyprowadzenia. Omówimy je w kolejnych lekcjach.



Rejestr **DDRx**, gdzie w miejsce x wpisujemy nazwę portu określa czy dany pin ma być wejściem czy wyjściem sygnału cyfrowego.

0 – wejście
1 – wyjście

W naszym przypadku chcieliśmy, aby PC2 było wyjściem sygnału. Zatem do DDRC w pozycji odpowiadającej pinowi PC2 wpisano wartość 1. Pozostałe piny nie miały niczym sterować, więc pozostały wejściami.

```
DDRC  = 0b00000100;
```

Przyporządkowując poszczególne cyferki do pinów wygląda to następująco:

```
          PC7 PC6 PC5 PC4 PC3 PC2 PC1 PC0
DDRC = 0b 0   0   0   0   0   1   0   0
```

Tutaj należy się kilka słów wyjaśnienia. W programowaniu mikrokontrolerów stosuje się trzy sposoby zapisu wartości liczbowej. Dziesiętny (DEC), szesnastkowy (HEX) oraz dwójkowy (BIN). W nawiasach znajdują się skróty angielskich nazw tych systemów. Odpowiednio decimal, hexadecimal, binary. Warto o tym wiedzieć, gdyż w literaturze nawet polskojęzycznej często spotyka się angielskie określenia. W powyższej linijce został zastosowany zapis dwójkowy ale tą samą instrukcję można zapisać stosując inny system kodowania.

```
DDRC = 0b00000100;  //zapis dwójkowy (BIN) oznaczany 0b

       lub

DDRC = 0x04;  //zapis szesnastkowy (HEX) oznaczany 0x

       lub

DDRC = 4;  //zapis dziesiętny (DEC) bez oznaczenia
```

Tym trzem systemom zapisu przyjrzymy się w dalszej części instrukcji. Nie jest to trudne a użytkownik zapewne sam dojdzie do wniosku, że system dziesiętny nie jest wygodny dla programisty.

Podsumowując ustawiliśmy pin PC2 jako wyjście. Teraz w rejestrze stanu wyjścia **PORTx** wpiszemy wartość jaka ma się pojawić na wyjściu. Do wyboru mamy wartość logiczną 0 lub 1.

0 – potencjał masy, czyli **GND**
1 – potencjał zasilania **VCC**, czyli **5V**

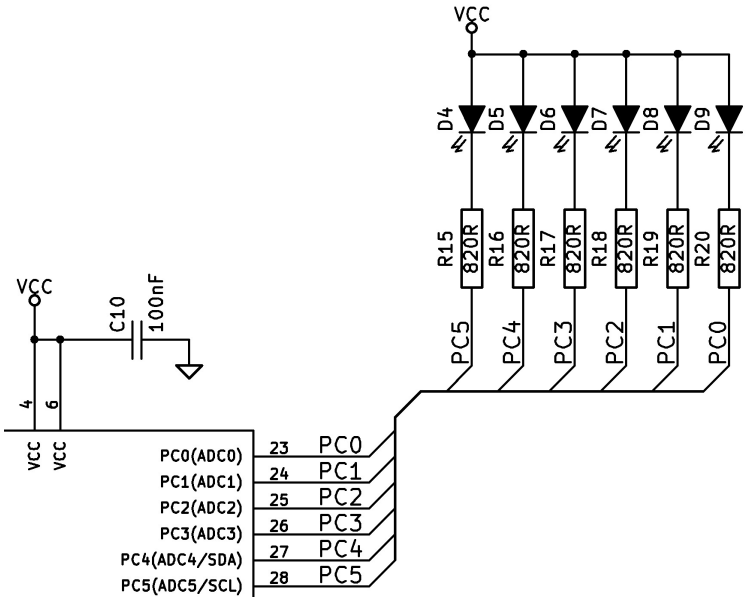
W naszym przykładzie wpisaliśmy:

```
PORTC = 0b11111011;
```

Znowu przyporządkowując poszczególne cyferki do pinów wygląda to następująco:

```
           PC7 PC6 PC5 PC4 PC3 PC2 PC1 PC0
PORTC = 0b 1   1   1   1   1   0   1   1
```

Dlaczego pin PC2 ustawiliśmy na 0 ? Przecież żeby zapalić diodę musi popłynąć przez nią prąd a my podłączyliśmy do PC2 masę (GND). W tym momencie należy spojrzeć na schemat. Jak widać na poniższym rysunku dioda D7 podłączona jest z jednej strony do zasilania VCC = 5V a z drugiej poprzez rezystor R18 do pinu PC2 mikrokontrolera. W takim przypadku, aby popłynął prąd wyjście PC2 musi stanowić masę, czyli potocznie mówiąc „minus" zasilania. Ustawienie wartości logicznej „1", czyli VCC na pinie PC2 zgasi diodę bo prąd nie popłynie z „plusa" do „plusa" zasilania.



Tym sposobem znamy już odpowiedź na pytanie jak zgasić diodę. Wystarczy podmienić wartość w PORTC.

```
PORTC = 0b11111011;  //zapala diodę D7

PORTC = 0b11111111;  //gasi diodę D7
```

Zanim zaczniemy mrugać diodą led spróbujmy zapalić wszystkie czerwone diody, czyli D4,D5,D6,D7,D8,D9. Myślę, że użytkownik już wie jak to zrobić ale dla zasady podaję przykład programu.

```
//Zapal wszystkie czerwone diody

DDRC  = 0b00111111;
PORTC = 0b11000000;
```

Gdy już wiemy jak zapalić i zgasić diody led, możemy pokusić się o wykonanie jakiegoś efektu świetlnego na diodach. Najprostszym będzie pulsowanie diody. Napiszemy teraz program, który będzie zapalał i gasił diodę D9 podłączoną do pinu PC0. Nasz program będzie miał postać.

```
DDRC  = 0b00000001;  //PC0 jako wyjście, reszta jako wejścia
while (1)  //wieczna pętla - w kółko wykonuje polecenia zawarte w nawiasie
{
  PORTC = 0b11111110;  //zapal diodę D9 (pin PC0)
  PORTC = 0b11111111;  //zgaś diodę D9 (pin PC0)
}
```

Kompilujemy program i ładujemy go do naszego zestawu. I jaki jest efekt ? Czy nasza dioda led mruga ? Zapewne świeci ale o mruganiu nie ma mowy. Otóż prawda jest taka, że ona mruga. Niestety robi to tak szybko, że nasze oczy tego nie widzą. Aby spowolnić ten proces musimy wprowadzić opóźnienia, czyli pomiędzy zapalaniem i gaszeniem diody musimy procesor czymś zająć. Najlepiej zlecić mu wykonywanie jakiejś pustej instrukcji powiedzmy 100 000 razy. Taką pustą instrukcją jest NOP. To polecenie nie realizuje żadnej konkretnej czynności, po prostu zajmuje czas procesora. Ten czas wynika z faktu, że procesor musi rozpoznać a następnie zinterpretować każdą napotkaną instrukcję nawet taką, która nie niesie dla niego żadnej informacji.
Linia opóźniająca powinna mieć postać:

```
for (x=0;x<100000;x++) asm (”nop”); //pętla opóźniająca
```

Gdzie zmienną „x” należy zadeklarować na początku programu.

```
unsigned long int x; //deklaracja zmiennej
```

Cały program powinien wyglądać następująco:

main.c

```
1   #include <avr/io.h>
2
3   int main(void)
4   {
5     unsigned long int x; //deklaracja zmiennej
6
7     DDRC  = 0b00000001; //PC0 jako wyjście
8     while (1) //wieczna pętla
9     {
10      PORTC = 0bllllllll0; //zapala diodę D9 (pin PC0)
11      for (x=0;x<100000;x++) asm ("nop"); //pętla opóźniająca
12
13      PORTC = 0bllllllll1; //gasi diodę D9 (pin PC0)
14      for (x=0;x<100000;x++) asm ("nop"); //pętla opóźniająca
15    }
16
17    return 0;
18  }
19
```

Teraz użytkownik będzie widział pulsującą diodę D9. Oczywiście efekt pulsowania można wykonać dla wszystkich diod led. Wystarczy zmienić wartość w DDRC oraz w pierwszej linijce ustawiającej PORTC. Jak widać kolejna linijka z użyciem PORTC pozostaje bez zmian.

```
DDRC  = 0b00111111;  //PC5..PC0 jako wyjścia
.
.
PORTC = 0b11000000;  //zapal diody D4..D9 (piny PC5..PC0)
.
.
PORTC = 0b11111111;  //zgaś diody D4..D9 (piny PC5..PC0)
.
.
```

Kompilator dostarcza bibliotekę **delay.h** w której znajduje się funkcja realizująca opóźnienie. Działa ona na podobnej zasadzie co nasza pętla „for" ale jej opóźnienie jest ściśle określone i odpowiada jednostce czasu podanej przez użytkownika. Funkcja bazuje na wartości zegara F_CPU zadeklarowanej w trakcie tworzenia projektu (patrz lekcja 2). Użytkownik wpisuje wartość opóźnienie np. 300ms a funkcja tworzy pętlę, która będzie wykonywana dokładnie przez taki czas. Aby móc skorzystać z tej funkcji należy dołączyć omawianą bibliotekę do programu. Wykonujemy to za pomocą poniższego polecenia.

```
#include <util/delay.h>
```

Funkcję opóźniającą wywołujemy pisząc:

_delay_ms(*czas w ms* **);**

Używając tej funkcji nasz poprzedni program będzie wyglądał następująco:

main.c

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3
4  int main(void)
5  {
6    DDRC  = 0b00000001; //PC0 jako wyjście
7    while (1) //wieczna pętla
8    {
9      PORTC = 0bllllllll0; //zapala diodę D9 (pin PC0)
10     _delay_ms(300); //opóźnienie 300ms = 0,3s
11
12     PORTC = 0bllllllll; //gasi diodę D9 (pin PC0)
13     _delay_ms(300); //opóźnienie 300ms = 0,3s
14   }
15
16   return 0;
17 }
18
```

Teraz wykonamy efekt „pływającego światełka”. Zadaniem programu będzie zapalanie i gaszenie kolejnych diod led w taki sposób, aby obserwator miał wrażenie, że światło się przemieszcza od prawej krawędzi do lewej i z powrotem. Tym razem najpierw pojawi się program a później jego omówienie. Zastosowany symbol sumy logicznej „|” znajduje się na klawiaturze nad Enterem.

main.c

```
1   #include <avr/io.h>
2   #include <util/delay.h>
3
4   int main(void)
5   {
6     unsigned char i;
7     unsigned char Diody;
8
9     DDRC  = 0b00111111; //PC5..PC0 jako wyjścia
10    Diody = 0b11111110; //na początek PC0=0
11    while (1) //wieczna pętla
12    {
13      for (i=0;i<5;i++) //wykonaj pętlę 5 razy
14      {
15        PORTC = Diody; //ustaw port zgodnie ze zmienną
16        _delay_ms(100); //opóźnienie 100ms
17        Diody = (Diody << 1) | 0b00000001;
18      }
19      for (i=0;i<5;i++) //wykonaj pętlę 5 razy
20      {
21        PORTC = Diody; //ustaw port zgodnie ze zmienną
22        _delay_ms(100); //opóźnienie 100ms
23        Diody = (Diody >> 1) | 0b10000000;
24      }
25    }
26
27    return 0;
28  }
29
```

W linijce 6 i 7 deklarujemy dwie zmienne bajtowe o nazwach „i” oraz „Diody”. Są to zmienne, które mogą przyjmować wartości od 0 do 255. Następnie ustawiamy DDRC w taki sposób, aby piny PC5..PC0 były wyjściami. Do zmiennej „Diody” wpisujemy liczbę odpowiadającą zapaleniu diody led podłączonej do pinu PC0. W wiecznej pętli tworzymy pętlę „for”, która ma być wykonana 5 razy. Zmienna „i” służy do liczenia okrążeń pętli. W linii 15 przepisujemy zawartość zmiennej „Diody” do rejestru PORTC. Na tym etapie jest to równoznaczne z instrukcją PORTC = 0b11111110, czyli zapaleniem diody D9. Następnie mamy 100 milisekundowe opóźnienie i operację matematyczną. Instrukcja z linii o numerze 17 wykonuje przesunięcie bitowe o jedną pozycje w lewo i ustawia 1 na najmłodszym bicie.

Czyli początkowo zmienna Diody = 0b11111110 a po instrukcji:

Diody = (Diody << 1) | 0b00000001;

będzie równa **0b11111101**

Działanie pierwszej pętli „for” wygląda następująco:

1 okrążenie pętli (i=0)

```
PORTC = Diody;  // Diody = 0b1111111 0 zapala się D9
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody << 1) | 0b00000001;
```

2 okrążenie pętli (i=1)

```
PORTC = Diody;  // Diody = 0b111111 0 1 zapala się D8
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody << 1) | 0b00000001;
```

3 okrążenie pętli (i=2)

```
PORTC = Diody;  // Diody = 0b11111 0 11 zapala się D7
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody << 1) | 0b00000001;
```

4 okrążenie pętli (i=3)

```
PORTC = Diody;  // Diody = 0b1111 0 111 zapala się D6
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody << 1) | 0b00000001;
```

5 okrążenie pętli (i=4) - ostatnie bo po nim nie będzie już spełniony warunek i < 5

```
PORTC = Diody;  // Diody = 0b111 0 1111 zapala się D5
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody << 1) | 0b00000001;
```

pętla kończy się ze zmienną Diody = 0b11**0**11111

Teraz rozpoczyna się druga pętla „for”, w której przesuwamy bity o jedną pozycję w prawo i dodatkowo ustawiamy wartość 1 na najstarszym bicie.

1 okrążenie pętli (i=0)

```
PORTC = Diody;  // Diody = 0b11 0 11111 zapala się D4
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody >> 1) | 0b10000000;
```

2 okrążenie pętli (i=1)

```
PORTC = Diody;  // Diody = 0b111 0 1111 zapala się D5
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody >> 1) | 0b10000000;
```

.
.
.

5 okrążenie pętli (i=4) - ostatnie bo po nim nie będzie już spełniony warunek i < 5

```
PORTC = Diody;  // Diody = 0b111111 0 1 zapala się D8
_delay_ms(100);  //opóźnienie 100ms
Diody = (Diody >> 1) | 0b10000000;
```

pętla kończy się ze zmienną Diody = 0b1111111**0**

Dioda D9 zostanie zapalona ponownie w pierwszej pętli „for”. Te dwie pętle będą wykonywane bez końca.

Przyszedł czas na omówienie systemów liczbowych. Przepisując powyższe programy bardzo łatwo się pomylić przy wartościach zapisanych w systemie dwójkowym. Owszem jest on najbardziej obrazowy i dlatego został wykorzystany w tej lekcji. Nie jest to jednak najwygodniejszy sposób zapisu liczby. W systemie dziesiętnym ryzyko błędu jest mniejsze ale w przypadku ustawienia pinów taka wartość niczego nie przypomina. Dla potwierdzenia tego faktu proponuję mały test.

Ustawiam port mikrokontrolera.

```
PORTC = 46;
```

Które piny będą miały wartość 0 ?

Prawda, że trudne pytanie ? Patrząc na tą liczbę nie jesteśmy w stanie odpowiedzieć. Teraz dla porównania tą samą liczbę zapiszemy w systemie dwójkowym.

```
PORTC = 0b00101110;  //to jest liczba 46
```

Chyba nikogo nie trzeba przekonywać, który system liczbowy w takim zadaniu jest lepszy. Ale pozostaje problem możliwości popełnienia błędu przy przepisywaniu tylu cyfr. Tutaj z pomocą przychodzi system szesnastkowy zwany także heksadecymalnym i oznaczany jako HEX. Ta sama liczba zapisana w systemie szesnastkowym wyglądałaby następująco.

```
PORTC = 0x2E;  //to jest liczba 46
```

Zapewne użytkownik nie widzi w tej chwili zalet systemu szesnastkowego bo przecież liczba 2E też niczego nie przypomina. Za chwilę wszystko się wyjaśni. Poniżej zapiszę liczby od 0 do 15 w trzech systemach liczbowych. Następnie pokażę jak łatwo wyobrazić sobie liczbę dwójkową patrząc na zapis szesnastkowy.

DEC	BIN	HEX
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Jak widać liczby od 0 do 9 w systemie szesnastkowym zapisywane są identycznie jak w systemie dziesiętnym. Dopiero liczby powyżej 9 zapisywane są literami. Jednak nie to jest najistotniejsze w naszym zastosowaniu. My potrzebujemy krótkiego zapisu liczby, który jednocześnie pozwoli nam wyobrazić sobie stany na poszczególnych pinach portu.

Teraz napiszę wartość 2E oraz jej odpowiednik w kodzie dwójkowym. Pytanie brzmi, czy widząc wartość szesnastkową byłbym w stanie samodzielnie napisać jej binarny odpowiednik ? Spróbujmy.

2E = 00101110

Myślę, że nadal będzie to trudne. Ale jeśli rozbiję liczbę spacją to wygląda to dużo prościej.

2E = 0010 1110

Czy czytelnik już zauważył o co chodzi ?

```
  2       E      (liczba szesnastkowa 2E)
0010    1110     (liczba dwójkowa 00101110)
```

Wystarczy zapamiętać zapis liczb od 0 do 15 w tych dwóch systemach, aby swobodnie dokonywać konwersji. Przy odrobinie wprawy programista patrząc na liczbę zapisaną w hex'ie wyobraża sobie jej binarny odpowiednik.

Przykłady:

Dwójkowy → Szesnastkowy

```
01011100        to inaczej   0101  1100
                               5     C    czyli = 5C (hex)

00110010        to inaczej   0011  0010 czy
                               3     2    czyli = 32 (hex)

10000100        to inaczej   1000  0100 czy
                               8     4    czyli = 84 (hex)
```

Szesnastkowy → Dwójkowy

```
7A              to inaczej     7     A
                             0111  1010    czyli = 01111010 (bin)

E0              to inaczej     E     0
                             1110  0000    czyli = 11100000 (bin)

B5              to inaczej     B     5
                             1011  0101    czyli = 10110101 (bin)
```