Politechnika Wrocławska

Podstawy Techniki Mikroprocesorowej
Laboratorium

Ćwiczenie 1
Porty I/O (we/wy)
Przerwania zewnętrzne

Program ćwiczenia:

- wprowadzenie do tematyki programowania mikrokontrolerów,
- podstawy programowania w asemblerze,
- obsługa portów we/wy,
- obsługa przerwań.

Zagadnienia do przygotowania:

- specyfika portów we/wy układu ATmega8535,
- rejestry uniwersalne – zastosowanie, wpisywanie danych,
- rejestry specjalne – zastosowanie, wpisywanie danych,
- skoki bezwarunkowe i warunkowe,
- wywoływanie podprogramów,
- inicjalizacja stosu,
- przerwania mikrokontrolera.

Literatura:

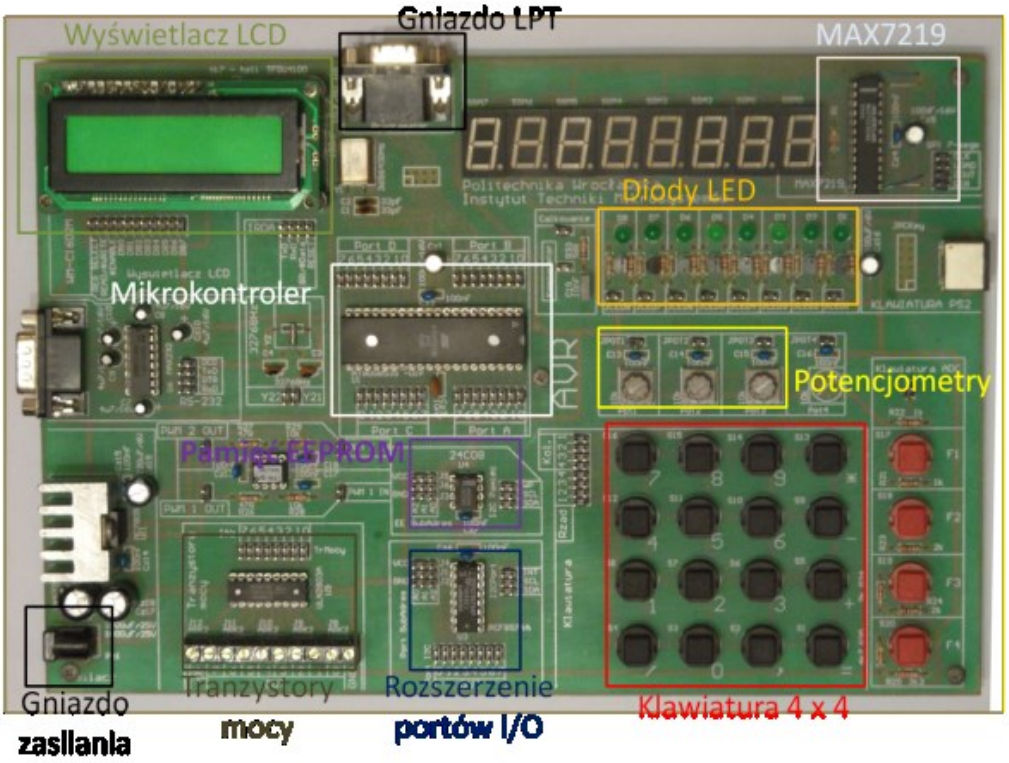
[1] Dokumentacja mikrokontrolera ATmega8535, www.atmel.com.
[2] Opis instrukcji mikrokontrolerów AVR: AVR Instruction Set Manual, www.atmel.com.
[3] Wykład.
[4] Mikrokontrolery AVR ATmega w praktyce, R. Baranowski, BTC 2005.

Zawartość instrukcji

UWAGA!
Do poprawnej obsługi urządzeń niezbędne jest korzystanie z dokumentacji mikrokontrolera.

1. Makiety dydaktyczne – informacje wstępne

Wszystkie ćwiczenia laboratoryjne realizowane są na makietach dydaktycznych (rys.1).:



Rys. 1. Widok makiety dydaktycznej.

Makieta dydaktyczna wyposażona jest między innymi w:

- mikrokontroler (ATmega8535 firmy ATMEL) – skrót MCU (MicroController Unit),
- osiem diod elektroluminescencyjnych (D0..D7),
- klawiaturę matrycową 4 × 4,
- potencjometry (Pot1, Pot2),
- wyświetlacz LCD (ze sterownikiem HD44780),
- 8 wyświetlaczy 7-segmentowych (sterowanych układem scalonym MAX7219),
- zewnętrzną pamięć EEPROM (układ scalony 24C08),
- rozszerzenie portów I/O (układ scalony PCF8574A),
- tranzystory mocy (układ scalony ULN2803A),
- konwerter RS232 / UART (MAX232).

- Wyprowadzenia we/wy mikrokontrolera (piny I/O, nóżki) są na makiecie podłączone do podwójnych listew kołkowych typu GOLDPIN. Listwy te podzielone są na cztery grupy – po jednej dla każdego z portów (A, B, C, D) mikrokontrolera.

- Diody elektroluminescencyjne (LED) sterowane są za pośrednictwem tranzystorów bipolarnych. Sygnały sterujące diodami należy doprowadzać do baz poszczególnych tranzystorów (złącza *JLEDx*, gdzie *x* odpowiada numerowi diody).
- Klawiatura matrycowa składa się z 16 przycisków podzielonych na 4 rzędy i 4 kolumny. Obsługiwana jest za pomocą podwójnej listwy kołkowej zawierającej 2 × 8 nóżek. Wybrany przycisk znajduje się na przecięciu podłączonego rzędu i kolumny.
- Potencjometry *Pot1*, *Pot2* podłącza się są za pomocą złącz *JPOT1* i *JPOT2*. Na wyprowadzeniu *JPOTx* występuje napięcie o wartości zależnej od aktualnej nastawy potencjometru (jest to wyjście sygnałowe potencjometru). Przy rezystancji potencjometru ustawionej na minimum wyprowadzenia *JPOTx* znajdują się na potencjale *Vcc = 5 V*.
- Wyświetlacz LCD umożliwia wyświetlanie znaków graficznych. Obsługiwany jest za pomocą 11 linii: 3 sterujących (*REG SELECT, READ/WRITE, ENABLE*) oraz 8 danych (*DB0..7*).
- Układ scalony MAX7219 służy do sterowania pracą wyświetlaczy 7-segmentowych. Programowanie układu odbywa się za pośrednictwem magistrali szeregowej *SPI*. Do sterowania pracą układu służą wyprowadzenia *CLK, LOAD, DOUT, DIN*.
- Tranzystory mocy służą do wzmacniania sygnałów pochodzących z mikrokontrolera w celu wysterowania np. uzwojeń silnika krokowego. Sygnał wejściowy należy podłączyć do odpowiednich nóżek podwójnej listwy kołkowej oznaczonej symbolem *IN0..7*. Sygnały wyjściowe buforów prądowych doprowadzone są do listwy zaciskowej oznaczonej symbolem *OUT0..7*.
- Zewnętrzna pamięć danych (EEPROM) umożliwia zapis i odczyt danych za pomocą magistrali *TWI (I^{2}C)*. Linie *A0..A2* (środkowe na listwie kołkowej) służą do ustalania adresu urządzenia (potrzebne w przypadku programowania więcej niż jednego układu pamięci). Linia *WP* służy do zabezpieczenia przed zapisem (gdy podłączona do *Vcc*, możliwy jest tylko odczyt). Sygnał zegarowy podłączany jest do linii *SCL*, a dane wysyłane/odbierane są przez linię *SDA*.
- Układ scalony PCF8574A umożliwia podłączenie dodatkowych portów I/O do mikrokontrolera. Układ programowany jest za pomocą magistrali *TWI*. Wyprowadzenia dodatkowego portu podłączone są do listwy kołkowej oznaczonej symbolem *Port I2C* (*OUT0..7I*). Do sterowania pracą układu służą linie *A0..2* (adres układu), *SCL* (sygnał zegarowy), *SDA* (wysyłanie/odbiór danych), *INT* (przerwanie od rozszerzonego portu).
- Konwerter RS232 / UART (układ scalony MAX232) umożliwia komunikację mikrokontrolera z komputerem PC. Po stronie mikrokontrolera wykorzystuje się magistralę UART, po stronie PC – port szeregowy i magistralę RS232. Układ MAX232 konwertuje poziomy stanów logicznych (pomiędzy 5 V a 12 V).

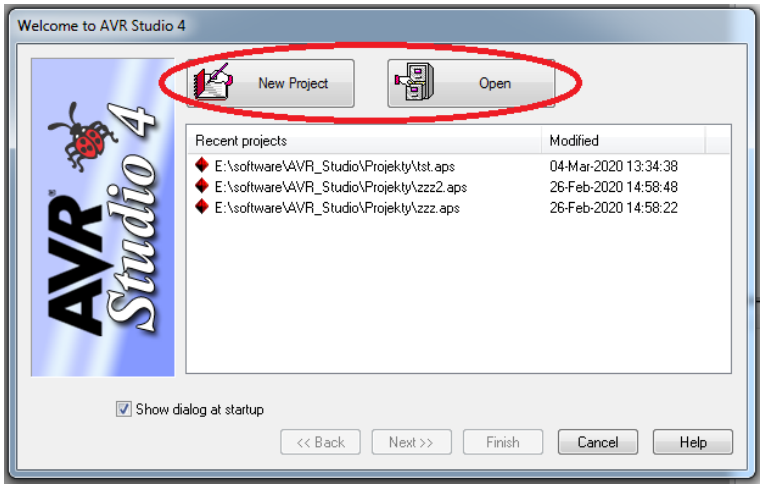
- Programowanie mikrokontrolera umieszczonego w makiecie odbywa się przy użyciu programatora USB dołączonego do zestawu. Programator należy podłączyć do portu PowerUSB zamocowanego na wewnętrznej ściance biurka.

2. Obsługa narzędzi programistycznych

Programowanie mikrokontrolera ATmega8535 odbywa się za pośrednictwem programów *AVR Studio 4.19* oraz *AVR8_Burn-O-Mat*. Oba są oprogramowaniem typu freeware i mogą być pobrane za darmo ze stron www.atmel.com (*AVR Studio*) oraz avr8-burn-o-mat.aaabbb.de (*AVR8_Burn-O-Mat*). *AVR Studio* służy do pisania kodu, jego asemblacji i symulacji działania mikrokontrolera. Kod w postaci binarnej wysyłany jest do MCU za pomocą *AVR8_Burn-O-Mat*.

2.1. Obsługa programu AVR Studio 4

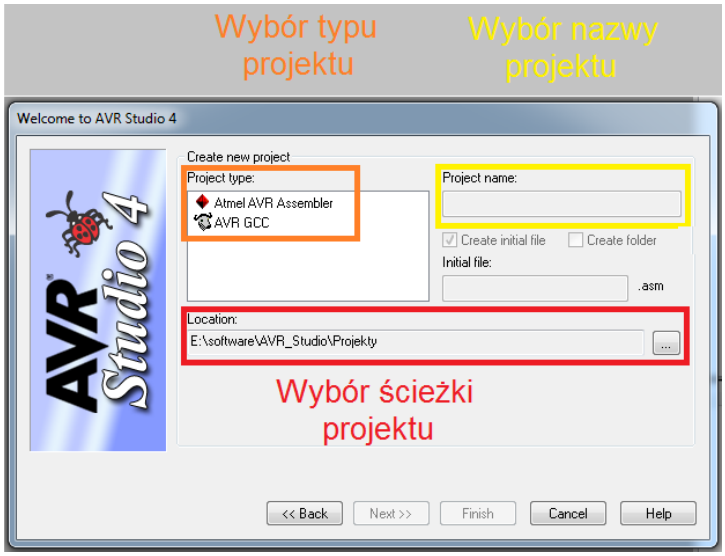
Po uruchomieniu programu *AVR Studio 4* wyświetlone zostanie okno powitalne (rys. 2). Jeżeli chcemy rozpocząć pracę nad nowym projektem należy wybrać przycisk *New Project*. Można również otworzyć jeden z wcześniej utworzonych projektów (przycisk *Open*).



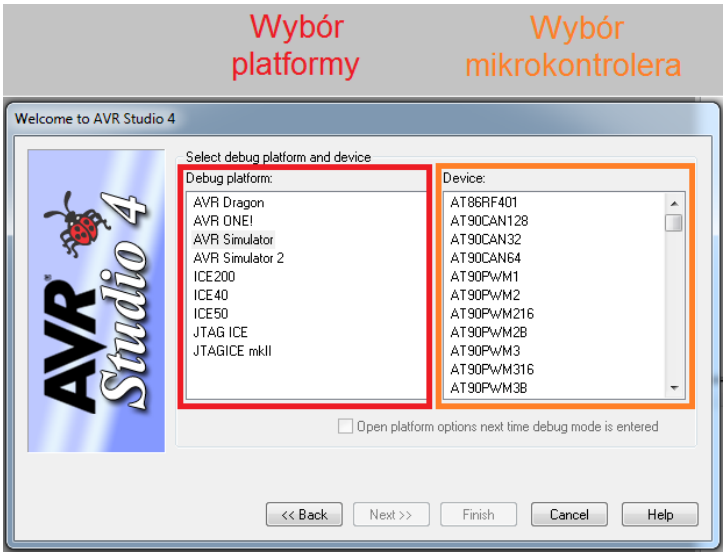
Rys. 2. Widok okna powitalnego programu *AVR Studio 4.*

Po wyborze nowego projektu, wyświetlone zostanie kolejne okno (rys. 3). Służy ono do wyboru typu, nazwy oraz ścieżki do projektu. Środowisko *AVR Studio 4* umożliwia programowanie mikrokontrolerów za pomocą języka *Assembler* (opcja *Atmel AVR Assembler*) lub wersji języka *C* zmodyfikowanej dla MCU (opcja *AVR GCC*). Program kursu przewiduje programowanie w *Assemblerze*. Po wyborze typu projektu, w polu po prawej stronie należy wpisać nazwę projektu (nie może zawierać spacji ani polskich liter). Następnie należy kliknąć przycisk z napisem *Next>>* (dolna część okna).

W kolejnym oknie (rys. 4) należy wybrać platformę debugowania (po lewej stronie, wybieramy *AVR Simulator*) oraz typ programowanego mikrokontrolera (po prawej, wybieramy *ATmega8535*). Następnie należy kliknąć przycisk z napisem *Finish*.

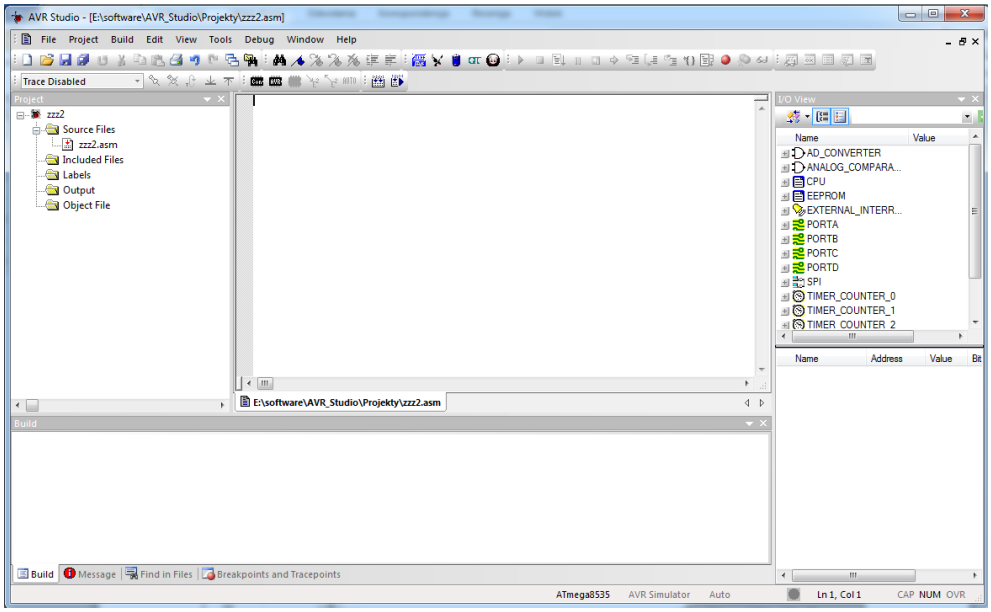


Rys. 3. Okno wyboru typu, nazwy oraz ścieżki nowego projektu.



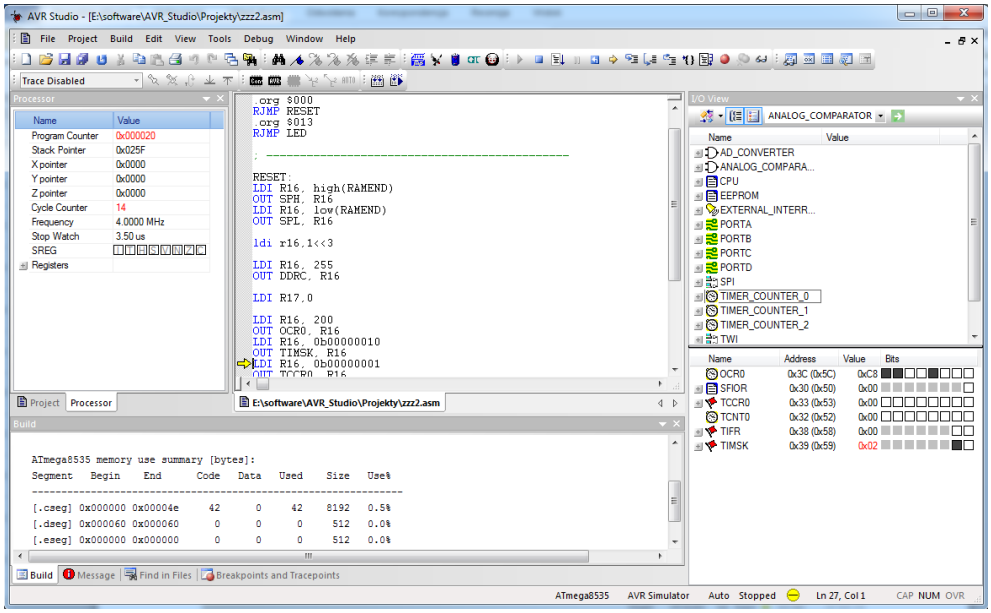
Rys. 4. Okno wyboru platformy debugowania oraz typu mikrokontrolera.

Na rys. 5 przedstawiono okno główne programu. Po lewej stronie znajduje się okno z zakładkami *Project* oraz *Processor*. Pierwsza zawiera informacje dotyczące systemu plików projektu. Druga zawiera wykaz najważniejszych rejestrów mikrokontrolera (np. licznik programu – *Program Counter*, wskaźnik wierzchołka stosu – *Stack Pointer,* rejestr statusu – *SREG* itd.) oraz rejestrów ogólnego przeznaczenia (*R0..31*). W oknie po prawej stronie znajduje się lista wszystkich *rejestrów I/O (rejestrów specjalnych)* mikrokontrolera. Kod programu wpisywany jest w centralnym polu. Instrukcje mikroprocesora (CPU) oznaczane są kolorem niebieskim, komentarze (rozpoczynają się znakiem ; lub // lub /* i */) kolorem zielonym, pozostały kod – kolor czarny.



Rys. 5. Widok okna głównego programu *AVR Studio 4.*

Po zakończeniu pisania kodu program należy poddać asemblacji. W tym celu należy kliknąć przycisk  (*Assemble*) lub nacisnąć klawisz F7. W wypadku gdy chcemy zasymulować działanie napisanego programu (tryb debugowania) należy kliknąć przycisk  (*Assemble and Run*) lub wcisnąć kombinację klawiszy Ctrl+F7. Podczas trwania symulacji (rys. 6) kolejna instrukcja, która będzie wykonana zaznaczona jest żółtą strzałką  znajdującą się po lewej stronie okna kodu programu (jest to graficzna reprezentacja aktualnej wartości licznika programu – *Program Counter*) .



Rys. 6. Widok okna głównego programu *AVR Studio 4* w trakcie debugowania (symulacji).

Aby przejść do kolejnej linijki kodu (symulacja wykonania instrukcji przez mikrokontroler) należy kliknąć ikonę *Step Into*  znajdującą się w górnym menu lub nacisnąć klawisz F11. W trakcie

trwania symulacji można przeglądać aktualne wartości wszystkich rejestrów mikrokontrolera na każdym etapie działania programu. Wartości zmienione w ostatnim kroku podświetlane są kolorem czerwonym.

Podstawowe ikony przydatne podczas symulacji:

 *Assemble* (F7) – asemblacja kodu;

 *Assemble and run* (Ctrl + F7) – asemblacja kodu i uruchomienie symulatora;

 *Start debugging* – uruchomienie symulatora (przydatne jeśli wcześniej wybrano F7);

 *Step Into (F11)* – symuluje wykonanie pojedynczej, kolejnej instrukcji;

 *Step Over (F10)* – symuluje wykonanie pojedynczej instrukcji lub całego podprogramu (gdy kolejna instrukcja to np. *RCALL*);

 *Step Out* – jeśli symulator jest wewnątrz podprogramu symuluje wykonanie wszystkich kolejnych instrukcji aż do jego zakończenia;

 *Run to Cursor* – należy ustawić kursor kilka (lub kilkanaście) linii kodu poniżej żółtej strzałki  ; symuluje wykonanie wszystkich instrukcji aż do kursora;

 *AutoStep* – symulacja automatyczna (nie trzeba naciskać np. F11);

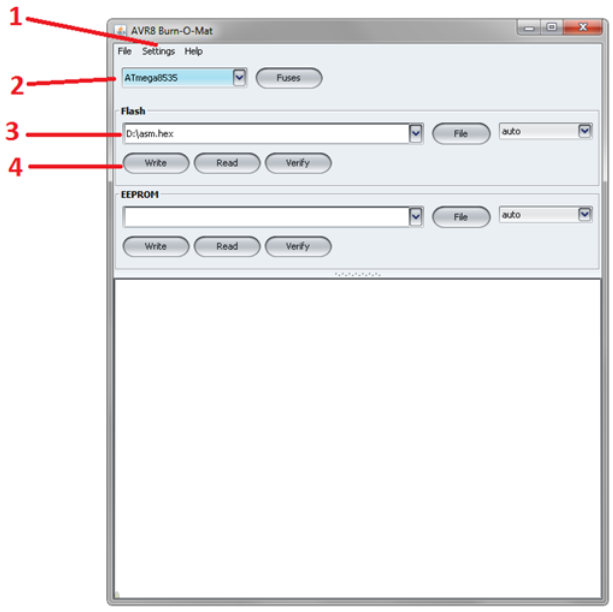
 *Break* – zatrzymuje symulację automatyczną;

 *Reset* – rozpoczęcie symulacji on nowa.

Po przeprowadzeniu asemblacji oprogramowanie *AVR Studio 4* generuje plik o takiej samej nazwie jak projekt i rozszerzeniu *hex*. Jest to plik wsadowy dla programu *AVR8_Burn-O-Mat.*

2.2. Obsługa programu AVR8_Burn-O-Mat

Wygenerowany plik *.hex* należy wybrać w programie *AVR8_Burn-O-Mat* w oknie (3) – jak na rys. 7. Przed wpisaniem programu do mikrokontrolera należy jeszcze sprawdzić czy w oknie (2) wybrano właściwy układ (ATmega8535). Wciśnięcie przycisku „Write” (4) zapisuje zawartość pliku *hex* w pamięci *FLASH EEPROM*.



Rys. 7. Widok okna głównego programu *AVR8_Burn-O-Mat.*

3. Programowanie portów I/O mikrokontrolera

Mikrokontroler ATmega8535 zawiera 64 rejestry specjalne (rejestry I/O) i 32 rejestry uniwersalne (Rx). Nazwy rejestrów specjalnych są związane z funkcją jaką te rejestry pełnią, bądź ze sprzętem którego pracę konfigurują (np. rejestr statusu *SREG,* rejestry *DDRx, PORTx, PINx* portów *I/O,*). Natomiast nazwy rejestrów uniwersalnych składają się z numeru rejestru (od 0 do 31) poprzedzonych literą *R* (np. *R16*). Aby móc korzystać ze zdefiniowanych nazw rejestrów należy program rozpocząć *dyrektywą asemblera* (zob. listing list. 1).

```
.include "m8535def.inc"
```

List. 1. Dyrektywa INCLUDE asemblera.

Rejestry specjalne służą do konfigurowania i sterowania pracą peryferii mikrokontrolera (np. portów, liczników, przetwornika analogowo-cyfrowego, itd.). Operacje arytmetyczno-logiczne mogą być wykonywane tylko na rejestrach uniwersalnych (*Rx*). Rejestry te są używane również do programowania (konfigurowania) rejestrów specjalnych. Praktycznie wszystkie rejestry mikrokontrolera *ATmega8535* są 8 bitowe (poza nielicznymi wyjątkami).

Jednym z podstawowych peryferii każdego mikrokontrolera są porty wejścia/wyjścia. Służą one między innymi do wysyłania i odbierania danych przez MCU. Mikrokontroler *ATmega8535* wyposażony jest w 4 porty (A, B, C, D). Do konfiguracji kierunku portu (wejście/wyjście) służy rejestr specjalny *DDRx (Data Direction Register,* gdzie *x* to nazwa portu np. dla portu A rejestr będzie nosił nazwę *DDRA*). Każdemu z bitów w rejestrze *DDRx* odpowiada fizyczna nóżka wejścia/wyjścia mikrokontrolera, znajdująca się na porcie *X*. Np. bit nr 3 w rejestrze *DDRC* (czyli PC3) odpowiada

nóżce podpisanej „*Port C 3*” na makiecie dydaktycznej. Aby skonfigurować daną nóżkę jako wyjście należy wpisać w odpowiednie miejsce w rejestrze *DDRC* wartość logiczną ′1′. Jeżeli wpisana zostanie wartość logiczna ′0′ to dana linia portu będzie skonfigurowana jako wejście. Do wpisywania wartości do rejestrów służy między innymi komenda `LDI` jednak nie można wykorzystać jej do bezpośredniego konfigurowania rejestrów specjalnych. Oznacza to, że zapis z list. 2 spowoduje błąd podczas asemblacji programu.

```
LDI DDRC, 0b11111111 ;błędny kod!
```

List. 2. Błędny kod – niewłaściwy rejestr dla instrukcji LDI.

Poprawnie konfiguracja portu odbywa się za pośrednictwem rejestru uniwersalnego z grupy *R16..31* (uwaga: instrukcja `LDI` nie współpracuje z rejestrami uniwersalnymi *R0..15)* – np. list. 3.

```
LDI R16, 0b11111111
OUT DDRC, R16
```

List. 3. Poprawny kod – ładowanie 0b11111111 do DDRC.

Użyta powyżej komenda `OUT` powoduje przepisanie wartości z rejestru uniwersalnego *R16* do specjalnego *DDRC*. Wykonanie tego fragmentu kodu spowoduje wysterowanie wszystkich 8 nóżek portu C (PC0..7) jako wyjścia.
Możliwe jest skonfigurowanie części nóżek portu jako wejścia, części jako wyjścia, np. list. 4.

```
LDI R16, 0b00011111
OUT DDRA, R16
```

List. 4. Konfiguracja rejestru DDRA.

Taki zapis spowoduje, że 3 najstarsze nóżki portu A (linie PA7..5) będą wejściami, natomiast pozostałe będą wyjściami (linie PA4..0).

Jeżeli dana nóżka (nóżki, cały port) pracuje jako **wejście** stany logiczne odpowiadające sygnałom dostarczonym z zewnątrz do linii portu są wpisywane przez mikrokontroler do rejestru *PINx*. Aby odczytać wartość z portu i zapisać ją do rejestru uniwersalnego należy użyć instrukcji IN, np. list. 5.

```
LDI R16, 0b00000000 //wpisz 00000000 do rejestru R16
OUT DDRA, R16       //cały port A ustawiony jako wejście
IN R17, PINA        //przepisanie wartości z rej. PINA do R17
```

List. 5. Kopiowanie zawartości rejestru PINA do R17.

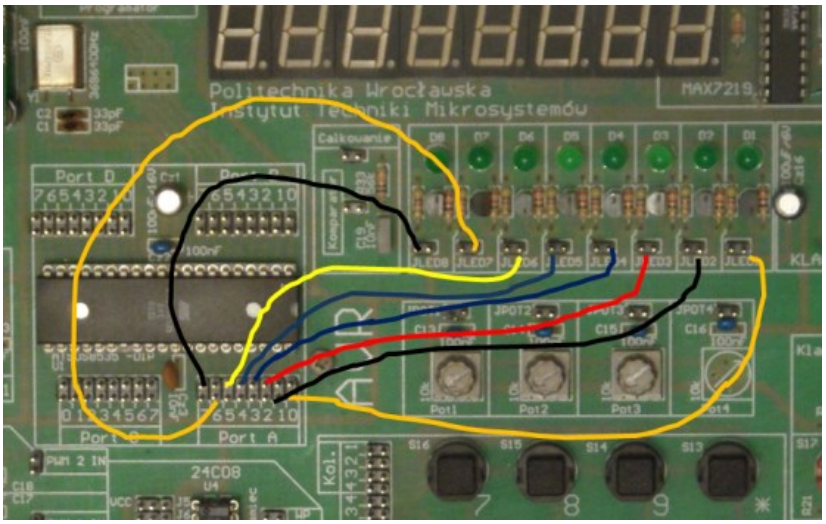
Jeżeli dana nóżka (nóżki, cały port) pracuje jako **wyjście** to wysyłane nią dane należy wpisywać do rejestru *PORTx*. W wypadku wysyłania danych np. portem A należy użyć rejestru *PORTA* (uwaga:

„port A” oznacza jedno z urządzeń wewnątrz mikrokontrolera, „*PORTA*” oznacza jeden z rejestrów sterujących portem A):

```
.include "m8535def.inc"
LDI R17, 0b11111111      //załaduj 11111111 do R16
OUT DDRA, R17            //cały port A ustawiony jako wyjście
LDI R17, 0b01010101      //załaduj 0b01010101 do R17
OUT PORTA, R17           //wyślij zawartość R17 portem A
```

List. 6. Wysłanie informacji 0b01010101 przy użyciu portu A.

Wpisanie powyższego programu do mikrokontrolera i podłączenie jego portu A z wejściami sterującymi pracą diod LED na makiecie (rys. 8) spowoduje, że zaświeci się co druga dioda (D1, D3, D5, D7), co druga nie będzie świeciła (D2, D4, D6, D8).



Rys. 8. Schemat podłączenia portu A mikrokontrolera z wejściami diod LED.

Poprawnie napisany program powinien zawierać na końcu jeszcze jedną instrukcję, która spowoduje jego zapętlenie. Uniemożliwi to mikroprocesorowi (CPU) przejścia do kolejnej linii w pamięci programu (następnej po `OUT PORTA, R17`). Wprawdzie ona, jak i kolejne nie zawierają już instrukcji, jednak mikroprocesor musi sprawdzić zawartość każdej z nich (4096 linii dla ATmega8535), co znacznie spowalnia działanie programu. Zapętlenie programu realizuje się za pomocą *etykiety* oraz instrukcji `RJMP` (wyjaśnienie użytego mechanizmu znajduje się w dalszej części instrukcji). Program powodujący cykliczne zaświecanie i gaszenie wszystkich diod LED może mieć postać jak na list. 7.

```
.include "m8535def.inc"

LDI R16, 0b11111111
LDI R17, 0
OUT DDRA, R16          //cały port A ustawiony jako wyjście

powtorz:               //etykieta – omówiona w punkcie 4
  OUT PORTA, R16       //wyślij zawartość R16 portem A => LED świecą
  NOP                  //No Operation – czas na ustalenie się stanu log.
  OUT PORTA, R17       //wyślij zawartość R17 portem A => LED gasną
RJMP powtorz           //zapętlenie programu – omówione w punkcie 4
```

List. 7. Program powodujący miganie diod LED.

Podsumowując:

- mikrokontroler ATmega8535 posiada 4 urządzenia typu ‘PORT’: portA, portB, portC, portD;
- do sterowania każdym z tych urządzeń służą 3 rejestry specjalne: *DDRx, PORTx* i *PINx*;
- bity w tych rejestrach oznaczamy w jednolity sposób: PC0, PB2, PA7, PD5, itp.

4. Programowanie pętli, skoków i podprogramów

Skoki bezwarunkowe

Mikrokontroler wykonuje program rozkaz po rozkazie. Aby przejść do komendy innej niż następna trzeba zastosować jedną z dostępnych instrukcji skoku. Wykonanie jej spowoduje przejście do wybranej linii kodu programu. Miejsce to można wskazać podając numer linijki kodu programu lub, co jest znacznie wygodniejsze, oznaczyć je etykietą. Etykieta może mieć prawie dowolną nazwę. Musi rozpoczynać się literą i kończyć dwukropkiem. W kodzie programu nie mogą występować dwie identyczne etykiety. List. 8 przedstawia przykład deklaracji etykiety oraz odwołania do niej.

```
zatrzymaj:      //deklaracja etykiety "zatrzymaj"
...
RJMP zatrzymaj  //skocz do etykiety "zatrzymaj"
```

List. 8. Przykład deklaracji etykiety ‘zatrzymaj’ oraz odwołania do niej.

Na potrzeby kursu można wyróżnić dwie instrukcje skoku bezwarunkowego: RJMP oraz RCALL. RJMP (skok bezwarunkowy bez śladu) jest operacją najczęściej wykorzystywaną przy programowaniu pętli i prostych skoków. RCALL (skok bezwarunkowy ze śladem) jest związany z podprogramami i będzie omówiony w dalszej części instrukcji.

Skoki warunkowe

Jeżeli przejście do wskazanej linii kodu programu (etykiety) ma odbywać się tylko w przypadku spełniania określonego warunku, przydatne są instrukcje skoku warunkowego. W mikrokontrolerach AVR występują 3 grupy takich operacji:

- instrukcje typu SKIP (4 instrukcje);
- instrukcja CPSE (1 instrukcja);
- instrukcje typu BRANCH (20 instrukcji).

Instrukcje warunkowe w obrębie każdej z grup działają niemal identycznie. W uproszczeniu można stwierdzić, że programista mikrokontrolerów AVR ma do dyspozycji 3 instrukcje warunkowe, występujące w kilku/kilkunastu wariantach.

Instrukcje typu SKIP testują stan bitu w rejestrze (list. 9). Instrukcja umieszczona bezpośrednio pod SKIP zostanie wykonana lub zignorowana, w zależności od wyników testu (dotyczy tylko jednej instrukcji, bezpośrednio pod SKIP). Do dyspozycji mamy 4 instrukcje tego typu:

SBRS – Skip if Bit in Register **Set**; działa na rejestrach uniwersalnych R0..R31;

SBRC – Skip if Bit in Register **Clear**; działa na rejestrach uniwersalnych R0..R31;

SBIS – Skip if Bit in I/O Register **Set**; działa na rejestrach specjalnych;

SBIC – Skip if Bit in I/O Register **Clear**; działa na rejestrach specjalnych.

```
LDI R17, 0b00001110 //logiczne '1' na bitach nr 1, 2 i 3

SBRS R17,3          //testowanie bitu nr 3 w R17; SBRS => skip if Set
  RJMP etykieta_1   //skok zostanie zignorowany, bo bit nr 3 = '1'

SBRC R17,1          //test bitu nr 1 w R17; SBRC => skip if Clear
  OUT PORTA,R17     //instrukcja zostanie wykonana, bo bit 1 = '1'

SBIC PINA,7         //testowanie bitu nr 7 w rejestrze spec. PINA
                    //SBIC => skip if Clear
  RJMP etykieta_2   /* z powyższego kodu nie wynika jaki jest stan bitu nr 7
                       w PINA, zatem nie można stwierdzić czy skok nastąpi */
RJMP etykieta_3     /* skok na pewno zostanie wykonany - bezpośrednio
                       przed RJMP nie ma instrukcji warunkowej */
```

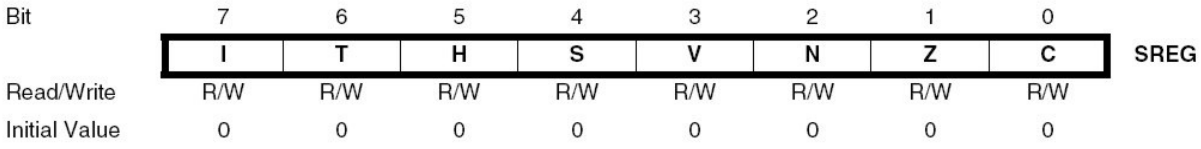
List. 9. Przykład użycia instrukcji warunkowych z grupy SKIP.

Instrukcja CPSE (ComPare Skip if Equal) działa na podobnej zasadzie (list. 10). Porównuje dwa rejestry uniwersalne. Jeżeli są sobie równe zostanie zignorowana instrukcja znajdująca się bezpośrednio pod CPSE.

```
CPSE R17, R18       //porównanie R17 z R18
    RJMP etykieta_1 ;jeżeli R17=R18 instrukcja zostanie zignorowana
RJMP etykieta_2     ;ta instrukcja na pewno zostanie wykonana

/* w efekcie: jeżeli R17=R18 nastąpi skok bez sladu do etykieta_2,
    w przeciwnym wypadku nastapi skok do etykieta_1 */
```

List. 10. Przykład użycia instrukcji warunkowej CPSE.

Instrukcje typu BRANCH powodują skok do wskazanej etykiety, gdy dany warunek jest spełniony. Badany warunek to zawsze stan flagi (bitu) w rejestrze specjalnym Status Register (SREG). W tym miejscu warto wspomnieć o funkcji SREG w mikroprocesorze. Jest to rejestr przechowujący istotne informacje o wyniku ostatniej operacji arytmetycznej lub logicznej wykonanej przez jednostkę arytmetyczno-logiczną (ALU) procesora (np. informacje o znaku liczby będącej wynikiem operacji, wyzerowaniu rejestru czy o jego przepełnieniu). Ze względu na dużą wagę przechowywanych informacji bity tego rejestru nazywane są flagami (rys. 9).



Rys. 9. Rejestr Status REGister (SREG).

Poszczególne flagi w SREG oznaczają:

I – flaga przerwań globalnych – niezwiązana z operacjami arytmetyczno-logicznymi;

T – bit magazynujący (sTorage) – niezwiązany z operacjami arytmetyczno-logicznymi;

H – Half carry – ustawiana (H = 1) gdy w wyniku operacji arytmetyczno-logicznej nastąpiło przeniesienie z bitu nr 3 na nr 4 (np. 0b00001111 + 0b00000001);

S – Sign – XOR flag N i V;

V – oVerflow – ustawiana (V = 1) gdy nastąpiła zmiana z 0b01111111 na 0b10000000
(bardzo istotna przy korzystaniu z kodu U2);

N – Negativ – ustawiana (N = 1) gdy najbardziej znaczący bit (MSB) = 1; kasowana gdy MSB = 0
(bardzo istotna przy korzystaniu z kodu U2);

Z – Zero – ustawiana (Z = 1) gdy wynik operacji arytmet. lub log. wynosi 0,

np.
```
LDI R16,1
DEC R16
```
lub
```
LDI R16, $FF
INC R16
```
lub
```
LDI R16, 128
LDI R17, $80
SUB R16, R17
```

C – Carry – ustawiana (C = 1) gdy nastąpiło przeniesienie z najstarszego bitu (oznacza przepełnienie 8-bitowego rejestru),

np.
```
LDI R16,1
ADIW R16,80
```
lub
```
LDI R16, $FF
INC R16
```
lub
```
LDI R17, $80
ROL R17
```

Przykładową instrukcją typu BRANCH jest *BREQ* (list. 11). Powoduje skok do miejsca oznaczonego etykietą, jeżeli w wyniku operacji wykonanej przez ALU rejestr uniwersalny został wyzerowany (jeżeli takie zdarzenie miało miejsce, ALU automatycznie ustawia flagę Z w rejestrze SREG, Z = '1').

```
LDI R16, 5          //R16 = 5
LDI R17, 3          //R17 = 3
DEC R17             //dekrementacja R17; R17 = 2, flaga Z = 0
BREQ gdzies_tam_1 //skoku nie będzie, bo Z = 0
DEC R17             //dekrementacja R17; R17 = 1, flaga Z = 0
DEC R17             //dekrementacja R17; R17 = 0, flaga Z = 1
BREQ gdzies_tam_2 //skok będzie, bo Z = 1
DEC R16             //dekrementacja R16, R16 = 4, flaga Z = 0
BREQ gdzies_tam_13 /* skoku nie będzie, bo Z = 0; nie ma znaczenia,
                      że R17 = 0; flaga dotyczy tylko ostatniej operacji*/
```

List. 11. Przykład użycia instrukcji warunkowych z grupy BRANCH.

Po wczytaniu instrukcji *BREQ* procesor automatycznie testuje wartość logiczną flagi Z. Jeżeli Z = '1' nastąpi skok do wskazanej etykiety. Jeżeli Z = '0' skoku nie będzie, procesor przejdzie do wykonania

następnej linii kodu. Dla CPU nie ma znaczenia, jaka operacja spowodowała Z = '1', ani na którym rejestrze uniwersalnym była wykonana. Dla programisty – ma to znaczenie kluczowe. Instrukcją warunkową komplementarną do *BREQ* jest *BRNE*, które również testuje flagę Z. W tym wypadku skok nastąpi, gdy Z = '0'. Dla każdej flagi w SREG istnieją 2 instrukcje typu BRANCH, które są komplementarne względem siebie. Ich wykaz zamieszczono w tab. 1, dostępny jest też w dokumentacji mikrokontrolera [1] i dokumencie AVR INSTRUCTION SET [2].

Tab. 1. Wykaz instrukcji skoku warunkowego BRANCH [2]

Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT(1)	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE(1)	Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO(1)	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/ BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH(1)	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

Note: Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd.

Uwaga: proszę zwrócić uwagę na ostatnią kolumnę. Informuje, czy instrukcje działają z kodem NKB (unsigned) czy U2 (signed).

Instrukcja *BRNE* może posłużyć do stworzenia pętli opóźniającej. Pętla taka może być wykorzystana np. w programie sterującym pracą diod LED na makiecie. W przedstawionym wcześniej programie (list. 7) diody będą mrugać z częstotliwością w przybliżeniu równą połowie częstotliwości taktowania procesora. W wypadku mikrokontrolera *ATmega8535* umieszczonego na makiecie jest ona na tyle wysoka (1 MHz), że niemożliwe jest zaobserwowanie migania. Zwiększenie czasu opóźnienia można osiągnąć przez uzupełnienie programu o pętlę opóźniającą, która może wyglądać na przykład tak, jak na list. 12.

```
LDI R20, 150      //R20 = 150
delay:            //deklaracja etykiety "delay"
DEC R20           //dekrementacja R20; R20 = 149; Z = 0
BRNE delay        //skocz do "delay" jeżeli R20 != 0 albo
                  //wyjdź z pętli jeżeli R20 = 0
```

List. 12. Przykładowa pętla opóźniająca.

Wykonanie tak skonstruowanej pętli opóźniającej zajmie mikrokontrolerowi około 3·150 cykli zegarowych (′3′ jest w tym wzorze stałą, 150 to wartość wpisana do R20 – można ją zmieniać w zakresie 0..255). Pętle należy umieścić dwukrotnie w kodzie – po instrukcjach zaświecających i gaszących diody LED.

- Oszacować: czy takie opóźnienie jest wystarczające, aby je zauważyć?
- Jeśli nie – zaproponować kod dla pętli zagnieżdżonej.

Jednak przygotowany w taki sposób program traci na przejrzystości. Dlatego lepszym rozwiązaniem jest zastosowanie podprogramu. W wypadku Asemblera podprogram musi rozpocząć się etykietą i kończyć instrukcją *RET*. Aby wywołać odpowiedni podprogram należy jego etykietę poprzedzić instrukcją *RCALL* (list. 13).

```
RCALL podprog_opoznienie     /* skok do miejsca oznaczonego
                                etykietą "podprog_opoznienie" */
```

List. 13. Wywołanie podprogramu.

Jednak, aby móc użyć instrukcji *RCALL* należy uprzednio dokonać inicjalizacji stosu w pamięci SRAM. Instrukcja *RCALL* (skok bezwarunkowy ze śladem) powoduje zapisanie w pamięci mikrokontrolera (na stosie) adresu powrotu (numeru linii kodu programu). Program wraca do punktu, z którego nastąpiło wywołanie podprogramu, gdy w podprogramie natrafi na komendę *RET*. Ponieważ adres powrotu zapisywany jest na stosie, niezbędne jest zainicjalizowanie wskaźnika wierzchołka stosu (*Stack Pointer)* – list. 14.

```
//inicjalizacja stosu
LDI R16,low(RAMEND)
OUT SPL, R16
LDI R16,high(RAMEND)
OUT SPH, R16
```

List. 13. Inicjalizacja stosu – przypisanie wartości wskaźnikowi stosu (Stack Pointer, SP).

RAMEND jest stałą zapisaną w pliku „m8535def.inc”, w której przechowywany jest adres wierzchołka stosu. Rejestry *SPL (Stack Pointer Low)* oraz *SPH (Stack Pointer High)* są rejestrami, do których zapisywana jest młodsza (low) oraz starsza (high) część adresu wskaźnika wierzchołka stosu. Ten fragment kodu powinien zawsze znajdować się na początku programu zaraz za dyrektywami Asemblera (instrukcje rozpoczynające się kropką, np. *.include*).

Zmodyfikowany kod programu powodujący cykliczne zaświecanie i gaszenie diod LED z wykorzystaniem podprogramów może mieć postać jak na list. 14.

```
.include "m8535def.inc"

LDI R16,low(RAMEND)
OUT SPL, R16
LDI R16,high(RAMEND)
OUT SPH, R16

LDI R16, 0b11111111
LDI R17, 0b00000000

OUT DDRA, R16        //ustaw nóżki portu A jako wyjściowe

powtorz:             // program główny
 OUT PORTA, R16      // zaświeć diody
  RCALL delay0       // wywołanie podprogramu "delay0"
 OUT PORTA, R17      // zgaś diody
  RCALL delay0
RJMP powtorz         // zapętlenie program głównego

delay0:              //podprogram z pętlą opóźniającą (~ 750 us)
  ldi R18, 250
  delay0_1:
   dec R18
  brne delay0_1
ret                  //powrót z podprogramu do programu głównego
```

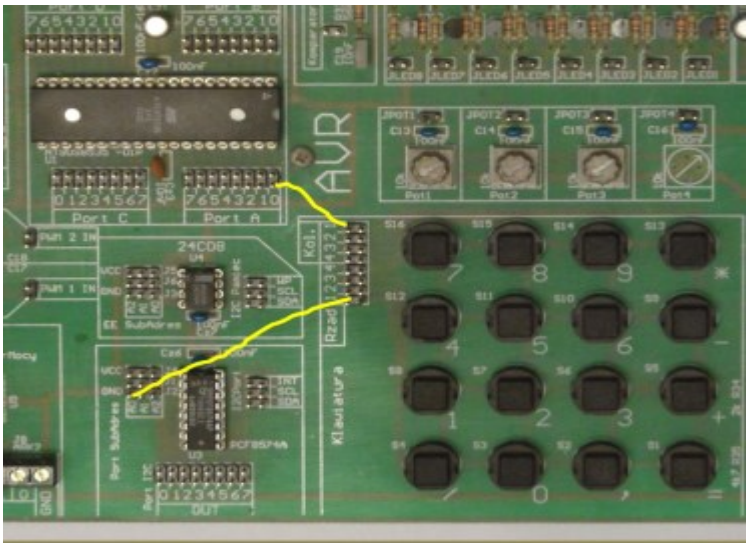
List. 14. Program powodujący miganie diod LED, z podprogramem opóźniającym.

- Oszacować: Czy takie opóźnienie jest wystarczające, aby je zauważyć?
- Jeśli nie – zaproponować kod dla pętli zagnieżdżonej.

5. Obsługa przycisków

Schemat podłączenia pojedynczego przycisku do mikrokontrolera przedstawiono na Rys. 10.

- Który przycisk został wybrany?



Rys. 10. Podłączenie przycisku do mikrokontrolera na makiecie dydaktycznej.

Port mikrokontrolera wysterowany jako wejście może zachowywać się niestabilnie. Jest to związane z indukowaniem się zakłóceń elektromagnetycznych. Każdy pin jak i ścieżki przewodzące na płytce PCB (makiecie) są swojego rodzaju antenami, podatnymi na sygnały np. z nadajników telefonów komórkowych, sieć Wi –Fi czy nawet na ładunki elektrostatyczne przenoszone na ciele człowieka. Taki ładunek, dostając się portem wejściowym do czułej logiki procesora, może zostać zinterpretowany jako rzeczywisty sygnał logiczny przysłany z innego układu, podłączonego do MCU. W efekcie program może zadziałać w nieodpowiedni sposób. Aby przed tym zabezpieczyć w liniach portów mikrokontrolera używanych jako wejścia koniecznie należy użyć mechanizmu *pull-up*. Jest to „podciągnięcie” (brak lepszego odpowiednika w języku polskim) stanu logicznego wybranych linii wejściowych do ′1′ (np. 5 V). Mechanizmu *pull-up* uruchamia się programowo, w sposób przedstawiony na list. 15.

```
LDI R16, 0
OUT DDRA, R16       //port A jako wejście
OUT DDRB, R16       //port B jako wejście
LDI R16, 0b11111111
OUT PORTA, R16      //pull-up na wszystkich liniach portu A
LDI R16, 0b11100000
OUT PORTB, R16      //pull-up na 3 najstarszych liniach portu B
```

List. 15. Programowe uruchomienie mechanizmu *pull-up*.

W tej sytuacji ustalone zostały wysokie stany logiczne na wszystkich liniach portu A (PA7..0) i możliwe jest wykrywanie przez mikrokontroler zewnętrznej zmiany stanu logicznego z ′1′ na ′0′. Od strony hardwarowej jedno wyprowadzenie przycisku należy podłączyć do pinu wejściowego mikrokontrolera, natomiast drugie do *GND* (stan logiczny ′0′). Przy właściwym podłączeniu wciśnięcie przycisku spowoduje ustalenie potencjału *GND* na odpowiedniej linii portu. Wykrycie przez mikrokontroler wciśnięcia przycisku realizuje się przez obserwację stanu logicznego na linii portu, do której podłączony jest przycisk.

```
//diodę podłączono do portu A, nóżka nr 7
//przycisk podłączono do portu A, nóżka nr 0

.include "m8535def.inc"

LDI R16,0b10000000
OUT DDRA, R16       //linia nr 7 ustawiona jako WY, pozostałe jako WE
LDI R17, 0b01111111
OUT PORTA, R17      // włączony pull-up na wysztkich wejściach (logiczne '1')
                    // na nóżce nr 7 logiczne '0' - dioda nie świeci
przycisk:
  SBIS PINA, 0      //sprawdź stan logiczny na nóżce nr 0 portu A - PINA(0)
   RJMP zaswiec     //skocz do "zaswiec" jeśli '0' (przycisk wciśnięty)
                    //zignoruj RJMP jeśli '1' (przycisk zwolniony)

  OUT PORTA,R17     //dioda nie świeci - wykona się, gdy nie nastąpi skok
RJMP przycisk       //zapętlenie programu głównego

zaswiec:            //fragment kodu wykonywany gdy wykryto wciśinięcie przycisku
  SBI PORTA,7       //ustaw '1' (zaświeć diodę) na PORTA(7)
RJMP przycisk       //powró do programu głównego

//jak zrealizować to z użyciem podprogramów (RCALL, RET)?
```

List. 16. Przykład obsługi przycisku.

Do wykrywania wciśnięcia przycisku można posłużyć się na przykład instrukcjami *SBIC, SBIS, SBRC, SBRS*, opisanymi w punkcie 4. Na list. 16 przedstawiono przykładowy fragment kodu powodującego zaświecenie diody LED gdy przycisk został wciśnięty i zgaszenie gdy został zwolniony. Opis instrukcji SBI dostępny jest w dokumentacji mikrokontrolera [1], dokumencie AVR INSTRUCTION SET [2] oraz notatkach do wykładu [3].

6. Przerwania zewnętrzne (INT0, INT1, INT2)

Przerwaniem nazywamy sygnał powodujący wstrzymanie aktualnie wykonywanego przez mikrokontroler programu i wykonanie podprogramu obsługi przerwania (ISR – Interrupt Service Routine). Po obsłużeniu przerwania mikrokontroler powraca do wykonywania poprzedniego programu w miejscu, w którym nastąpiło przerwanie.

Przerwania dzielimy na zewnętrzne oraz wewnętrzne. W pierwszym przypadku sygnał przerwania pochodzi od zewnętrznego urządzenia podłączonego do odpowiednich pinów mikrokontrolera. Dla mikrokontrolera *ATmega8535* sygnał przerwania zewnętrznego powinien być podłączony do jednej z trzech linii:

- Port D, nóżka 2 (PD2) – przerwanie *INT0*, adres $001;
- Port D, nóżka 3 (PD3) – przerwanie *INT1*, adres $002;
- Port B, nóżka 2 (PB2) – przerwanie *INT2*, adres $012.

Przerwania wewnętrzne mogą być wywoływane przez wybrane peryferia mikrokontrolera (np. licznik, przetwornik analogowy-cyfrowy, itd.).

W wypadku równoczesnego (w tym samym cyklu zegarowym) wystąpienia dwóch (lub więcej) przerwań obsługiwane jest przerwanie o wyższym priorytecie. Wykaz wszystkich przerwań oraz ich priorytetów można znaleźć w Tab. 19 (s. 46) dokumentacji układu *ATmega8535* [1]. We wspomnianej tabeli można odszukać również adres (nr linii w pamięci programu), pod którym należy umieścić podprogram obsługi przerwania. Na przykład, w wypadku przerwania *RESET* program przejdzie do instrukcji umieszczonej pod adresem *0x000* (czyli na początek programu). W wypadku wywołania przerwania INT2 – do linijki *0x012*. Algorytm obsługi przerwania przez CPU można przedstawić w uproszczony sposób następująco:

- nadejście żądania przerwania (np. wciśnięcie przycisku na linii INT1);
- CPU sprawdza, czy przerwania zostały odblokowane globalnie;
- CPU sprawdza, czy przerwanie INT1 zostało odblokowane;
- CPU przechodzi do adresu (linii kodu programu) związanej z danym przerwaniem (np. $002);
- programista powinien umieścić tam odnośnik do właściwego podprogramu – np. *RJMP skok*;

- CPU odkłada adres powrotu z ISR na wierzchołek stosu (zob. podprogramy) i wykonuje ISR;
- podprogram powinna kończyć instrukcja *RETi* – CPU pobiera adres powrotu i powraca do wykonywania przerwanego kodu.

W wypadku gdy w programie przewidujemy występowanie przerwań niezbędne jest uporządkowanie jego struktury za pomocą dyrektywy *.ORG*. Dyrektywa ta umożliwia nadanie konkretnego adresu występującej po niej linijce kodu, list. 17. Stosując dyrektywę *.ORG* należy pamiętać, że nie można użyć dwa razy tego samego adresu w jednym programie (2 linie nie mogą mieć jednakowego adresu). Adresy muszą być ułożone rosnąco:

ŹLE

```
.org $013
rjmp etykieta1
.org $000
rjmp etykieta2
```

DOBRZE

```
.org $000
rjmp etykieta2
.org $013
rjmp etykieta1
```

```
.org $000            ;ta linijka ma adres $000 (czyli 0)
  RJMP przerwanie1

.org $01             //ta linijka ma adres $01 (czyli 1)
  RJMP przerwanie2

.org $0000A          //ta linijka ma adres $0000A (czyli 10)
  RJMP przerwanie3

.org 0x013           //ta linijka ma adres $013 (czyli 19)
  RJMP przerwanie4

//liczba zer po '$' nie ma znaczenia
//przedrostki '$' oraz '0x' można stosować zamiennie
```

List. 17. Dyrektywa *.ORG* – porządkowanie struktury programu.

Aby móc korzystać z przerwań zewnętrznych należy:

1) wybrać sposób ich reakcji (sygnał wyzwalający);
2) odblokować wybrane przerwania w rejestrze GICR;
3) odblokować przerwania globalnie w rejestrze SREG.

Pierwszym krokiem jest wybór sygnału wyzwalającego przerwanie zewnętrzne. W tym celu należy skonfigurować rejestr *MCUCR (MCUCSR* w przypadku *INT2* – znajdź w dokumentacji), rys. 11.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Rys. 11. MCU Control Register – MCUCR.

Bity *ISC11* oraz *ISC10* używane są do konfigurowania sygnału wyzwalającego przerwania *INT1*, a bity *ISC01* oraz *ISC00* dla przerwania *INT0*. Sygnały wyzwalające przerwanie zewnętrzne oraz odpowiadające im kombinacje bitów *ISCx0* i *ISCx1* przedstawiono poniżej:

Tab. 2. Sygnały wyzwalające INT0 i INT1 (dla INT2 odpowiednia tabelka w dokumentacji ATmega8535).

ISCx1	***ISCx0***	**Sygnał wyzwalający**
0	0	Niski poziom na linii INT1/0
0	1	Zmiana stanu logicznego na linii INT1/0
1	0	Opadające zbocze na linii INT1/0
1	1	Narastające zbocze na linii INT1/0

Drugim krokiem jest odblokowanie wybranego przerwania przez ustawienie właściwego bitu w rejestrze *GICR (General Interrupt Control Register)*. W przypadku przerwań zewnętrznych *INT0, INT1* i *INT2* są to odpowiednio bity 6, 7 i 5 – rys. 12.

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Rys. 12. General Interrupt Control Register – GICR.

Po skonfigurowaniu rejestrów *GICR* oraz *MCUCR* należy odblokować przerwania globalnie. Odbywa się to przez ustawieniu 7 bitu (*Interrupt*) w rejestrze *SREG* – zob. rys. 9. Można to zrobić na kilka sposobów – list. 18.

```
SEI                 //ustaw flagę I w SREG
//LUB
sbi SREG, 7         //ustaw bit 7 w SREG
//LUB
...
```

List. 18. Globalne odblokowanie przerwań – przykładowo.

Powrót z programu obsługi przerwania następuję przez użycie instrukcji *RETI*. Do prawidłowego działania programu wykorzystującego przerwania niezbędna jest inicjalizacja stosu (zob. podprogramy).

Wyprowadzenia portu (D lub B), do których dostarczany jest sygnał przerwania mogą być skonfigurowane zarówno jako wejściowe (wówczas należy uruchomić *pull-up*) lub wyjściowe. Przykładowy program wykorzystujący przerwanie zewnętrzne INT2 przedstawiono na list. 19.

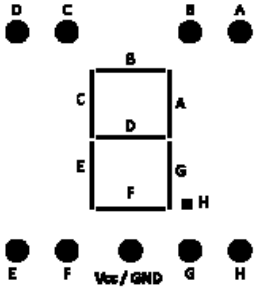
```
.include "m8535def.inc"
.org $000                       // adres RESET
  rjmp start
.org $012                       // adres przerwania INT2
  rjmp przerw2

start:
ldi R16, low(RAMEND)
out SPL, R16
ldi R16, high(RAMEND)
out SPH, R16

ldi R16,0b00000000
out DDRB, R16             // port B jest wejściem
ldi R16,0b00000100
out PORTB, R16            // pull-up na PB(2)

ldi R16, 0b00000000       // przerwanie INT2 będzie reagować na
out MCUCSR, R16           // poziom niski na linii PB(2)

ldi R16, 0b00100000       // odblokowanie przerwania INT2
out GICR, R16
sei                       // globalne odblokowanie przerwań

prog_glowny:
  ?                       // program główny
rjmp prog_glowny

przerw2:
  ?                       // program obsługi przerwania INT2
reti
```

List. 19. Przykładowy program wykorzystujący przerwanie zewnętrzne INT2.

7. Urządzenia wspomagające zajęcia laboratoryjne

7.1. Wyświetlacz 7-segmentowy (zewnętrzny)

Wyświetlacz ten służy do wyświetlania liczb dziesiętnych oraz niektórych liter. Składa się z siedmiu segmentów odpowiedzialnych za znak oraz jednego będącego kropką. Wyposażony jest w wyprowadzenia zapewniające zasilanie oraz umożliwiające sterowanie każdym segmentem niezależenie. Wyświetlacz używany podczas zajęć laboratoryjnych posiada 9 wyprowadzeń. Jest umieszczony na niewielkiej płytce PCB, dołączany do makiety za pośrednictwem przewodów. Schemat wyświetlacza siedmiosegmentowego przedstawiono na rys. 13.



Rys. 13. Schemat wyświetlacza siedmiosegmentowego.

Do obsługi wyświetlacza 7-segmentowego wystarcza jeden port 8-bitowy. Każdym segmentem steruje się indywidualnie, przy pomocy jednej nóżki wyjściowej MCU. Jeżeli wyświetlacz jest typu *wspólna katoda* (WK) jego dziewiątą nóżkę (środkowa w dolnym rzędzie, zob. rys. 13) należy podłączyć do *GND*. Jeżeli jest typu *wspólna anoda* (WA) – do *Vcc*. Dla układu WK podanie stanu wysokiego (′1′) na nóżkę wyświetlacza spowoduje zaświecenie wybranego segmentu, a stanu niskiego (′0′) jego zgaszenie. W wyświetlaczach WA jest odwrotnie. Na list. 20 zamieszczono przykładowy kod powodujący cykliczne zaświecanie i gaszenie wszystkich segmentów wyświetlacza siedmiosegmentowego.

```
.......              // dyrektywa .include, stos

ldi R16, $FF
out DDRA, R16        // cały port A ustawiony jako wyjście

zapal:
  out PORTA, R16     // wysyłanie '1' do segmentów wyświetlacza WK
  rcall delay0       // opóźnienie
  zgas:
    ldi R17, 0b00000000
    out PORTA, R17   // wysyłanie '0' do segmentów
    rcall delay0
rjmp zapal           // koniec programu głównego

delay0:              // podprogram z pętlą opóźniającą
;
ret
```

List. 19. Przykładowy program powodujący cykliczne zaświecanie i gaszenie wszystkich segmentów wyświetlacza 7-segmentowego.

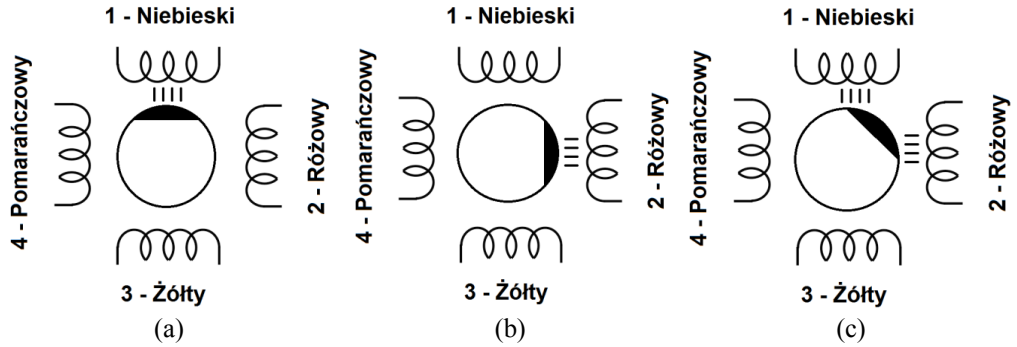
7.2. Silnik krokowy

W unipolarnym silniku krokowym prąd będzie przepływał przez cewki silnika zawsze w jednym kierunku (w odróżnieniu od silnika bipolarnego). Na rys. 14 przedstawiono uproszczony schemat takiego silnika. W podstawowym trybie pracy (sterowanie pełnokrokowe jednofazowe) w danej chwili prąd przepływa tylko przez jedną z 4 cewek. Rotor silnika ustawia się w jej kierunku (rys. 14a). Następnie pierwsza cewka jest odłączana a zasilanie przełączone do uzwojenia 2 – rotor wykonuje krok w prawą stronę (rys. 14b). Cewki zasilane są po kolei (4 kroki):

1 – 2 – 3 – 4 – 1 – .... kroki w prawą stronę, lub

4 – 3 – 2 – 1 – 4 – .... kroki w lewą stronę.

W omawianym, uproszczonym modelu pełny obrót rotora to 4 kroki. W silniku wykorzystywanym na zajęciach laboratoryjnych jest ich dużo więcej, dzięki odpowiedniej przekładni.



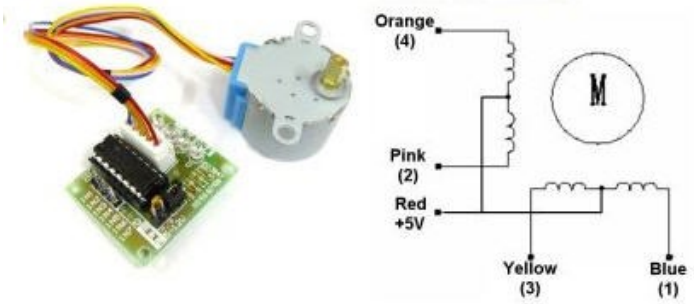
Rys. 14. Uproszczony schemat działania unipolarnego silnika krokowego: a) sterowanie pełnokrokowe jednofazowe, zasilanie uzwojenia cewki nr 1 (przewód niebieski), b) sterowanie pełnokrokowe jednofazowe, zasilanie uzwojenia cewki nr 2 (przewód różowy), c) sterowanie półkrokowe, jednoczesne zasilanie uzwojeń cewek nr 1 i 2.

Innym powszechnie stosowanym rodzajem pracy silnika krokowego unipolarnego jest sterowanie półkrokowe. Rotor zatrzymuje się również w punktach pośrednich, między dwoma cewkami (rys. 14c). Dzieje się tak, ponieważ między krokami opisanymi wcześniej pojawiają się dodatkowe, w których zasilane są jednocześnie dwie cewki (np. nr 1 oraz 2). W rezultacie pełny obrót omawianego, uproszczonego silnika wymaga 8 kroków. Cewki zasilane są w następującym porządku:

1 – (1+2) – 2 – (2+3) – 3 – (3+4) – 4 – (4+1) – .... kroki w prawą stronę, lub

4 – (4+3) – 3 – (3+2) – 2 – (2+1) – 1 – (1+4) – .... kroki w lewą stronę.

Sterowanie półkrokowe dwukrotnie zwiększa liczbę kroków przypadających na pełny obrót rotora.



Half-Step Switching Sequence

Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 Orange	-	-						-
3 Yellow		-	-	-				
2 Pink				-	-	-		
1 Blue						-	-	-

Rys. 15. Silnik krokowy wykorzystywany podczas zajęć laboratoryjnych: zdjęcie silnika oraz sterownika, schemat uzwojeń/wyprowadzeń, tabela zasilania kolejnych cewek dla obrotu zgodnego z ruchem wskazówek zegara (CW) – zamiast „-” wstaw ‘0’.

Na rys. 15 przedstawiono m.in. schemat uzwojeń i wyprowadzeń wykorzystywanego silnika. Cztery przewody (niebieski, różowy, żółty, pomarańczowy – jak na rys. 14) połączone są z końcami

dwóch uzwojeń tworzących cztery cewki (1, 2, 3, 4). Przewód czerwony to wspólne zasilanie (5 VDC). Jeżeli na wszystkich wyprowadzeniach 1÷4 podłączone zostanie napięcie 5 V, prąd nie będzie przepływał przez żadną z cewek. Jeżeli natomiast jedna z nich zostanie połączona z masą, prąd zacznie płynąć między zasilaniem (czerwony) a tym wyprowadzeniem – silnik ustawi się w określonej pozycji. Sterowanie obrotem silnika będzie polegało na cyklicznym wysyłaniu logicznego '0' na wyprowadzenie 1÷4 (lub ich parę). Dzięki temu w tym uzwojeniu (lub parze) pojawi się prąd elektryczny.

Do silnika dołączona jest płytka ze sterownikiem (tranzystory mocy) umożliwiająca odpowiednie wzmocnienie sygnałów logicznych generowanych przez mikrokontroler tak, aby moc była wystarczająca do obrotu rotora. Wyprowadzenia IN1÷IN4 odpowiadają przewodom 1÷4 (niebieski÷pomarańczowy) i powinny zostać połączone z wyjściami mikrokontrolera. Na diodach LED A, B, C, D można na bieżąco obserwować, w której fazie sekwencji kroków aktualnie znajduje się silnik (dioda A – zasilana cewka nr 1, B – cewka nr 2, itd.).

W trakcie zajęć laboratoryjnych należy napisać program sterujący obrotem silnika krokowego (podawanie odpowiedniej sekwencji kroków) oraz jego prędkością.

Prawidłowe podłączenie sterownika silnika krokowego do makiety laboratoryjnej:

- Sterownik wyposażony jest w 6 nóżek.
- Nóżki IN1÷IN4 należy podłączyć do wyjść mikrokontrolera. Wysłanie logicznego '0' na dowolny z nich spowoduje zasilenie danej cewki.
- Dwie nóżki zasilania, podpisane „- + 5-12V" należy połączyć z ich odpowiednikami na makiecie laboratoryjnej: „+" z *Vcc* oraz „-" z *GND*.
- Pozostałe dwie nóżki powinny pozostać połączone zworką.

8. Przykładowe zadania

1) Stworzyć cyfrowy sekundnik – co 1 s liczba binarna wyświetlana na diodach zwiększa się o 1.

Wskazówka: zmodyfikować pętlę opóźniającą w taki sposób, aby trwała 1.000.000 cykli zegarowych (dla taktowania 1 MHz)

2) Napisać program wyświetlający nr indeksu na wyświetlaczu siedmiosegmentowym (cyfra po cyfrze).

Wskazówka: po każdej cyfrze – pętla opóźniająca.

3) Napisać program obsługujący 4 przyciski. Po wciśnięciu każdego z przycisków na wyświetlaczu siedmiosegmentowym pojawi się inna cyfra/litera.

4) Napisać program, który będzie zliczał ile razy został wciśnięty przycisk. Wynik zliczania będzie wyświetlany na:
   a. linijce diod,
   b. wyświetlaczu siedmiosegmentowym.

Wskazówka: wykorzystać komendy *ROL/ROR*, *MOV*.

5) Napisać program, który zaświeci jedną z ośmiu diod. Po wciśnięciu jednego przycisku zaświecona dioda „przesunie się" w lewą stronę, a po wciśnięciu drugiego przycisku w prawo. Program zrealizować wykorzystując:
   a. instrukcje *SBIC/SBIS* lub *SBRC/SBRS,*
   b. przerwania zewnętrzne *INT0*, *INT1*.

Wskazówka: wykorzystać polecenia *ROL*, *ROR*

9. Zagadnienia do przygotowania

1) Rejestry uniwersalna i specjalne – podobieństwa, różnice, zastosowanie.
2) Rejestry specjalne DDRx, PORTx, PINx – zastosowanie, sposób użycia.
3) Rejestry specjalne SPL, SPH, GICR, GIFR, MCUCR, SREG – zastosowanie, sposób użycia.
4) Instrukcje LDI, IN, OUT, RJMP, BRNE – zastosowanie, sposób użycia.
5) Instrukcje SBIS, SBIC, RCALL, RET – zastosowanie, sposób użycia.
6) Stos – definicja, zastosowanie, sposób inicjalizacji.
7) Podprogramy – definicje, działanie, sposoby wywoływania.
8) Przerwania, przerwania zewnętrzne – definicja, rodzaje, sposób uruchomienia i obsługi (niezbędne elementy kodu programu).