Akademia
Górniczo-Hutnicza
w Krakowie
Instytut Elektroniki

Laboratorium mikrokontrolerów

Ćwiczenie 2

Pętle i instrukcje kontroli przepływu programu

Autor: Paweł Russek & Sebastian Koryciak

wer. 12.03.21

1. Wstęp

1.1. Cel ćwiczenia

Głównym celem instrukcji jest zapoznanie się z instrukcjami asemblera, które kontrolują przebieg programów w mikroprocesorze. Instrukcje te pozwalają na implementację pętli programowych oraz rozgałęzień warunkowych. W dalszej części laboratorium pokazany zostanie sposób realizacji podprogramów w kodzie procesora AVR.

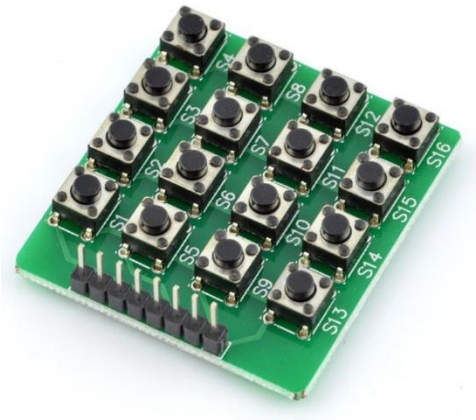
1.2. Konieczne wiadomości wstępne

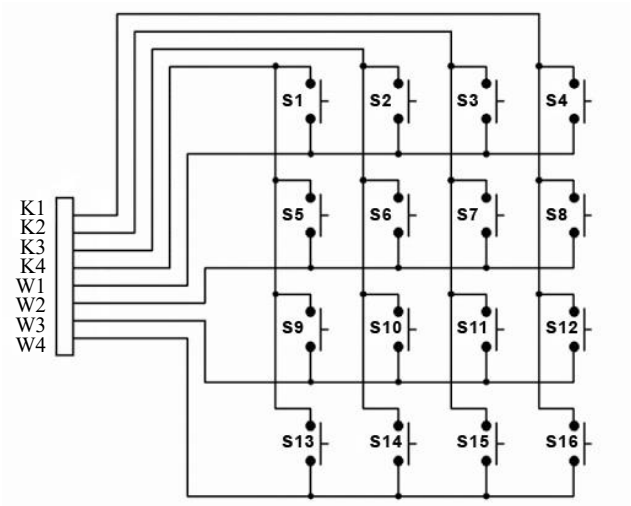
- Wykonanie instrukcji 1 z laboratorium techniki mikroprocesorowej.

2. Przygotowanie zestawu do ćwiczeń

2.1. Moduł klawiatury matrycowej

Matryca złożona jest z 16 przycisków typu tact switch rozłożonych w czterech wierszach i czterech kolumnach. Sygnały wyprowadzone są na 8 popularnych złączach goldpin - raster 2,54 mm. Jeżeli określony przycisk jest naciśnięty, to wtedy odpowiadająca mu kolumna (K1-K4) połączona jest z odpowiadającym mu wierszem (W1-W4).





Odpowiednie podłączenie matrycy umożliwia zmianę funkcjonalności klawiatury. Podłączenie jednego wiersza do masy (GND) powoduje, że funkcjonalność jest ograniczona do obsługi czterech przycisków (np. podłączenie wiersza W1 do masy i podłączenie wszystkich kolumn aktywuje przyciski S1, S2, S3 i S4). W takim przypadku naciśnięty przycisk wymusza stan niski na sygnałach odpowiadającym liniom kolumn (K1..K4).

Ważne

Moduł klawiatury nie posiada zewnętrznych rezystorów podciągających. Dlatego wymagane jest użycie w tym celu wewnętrznych rezystorów procesora AVR, które można aktywować w przypadku użycia danego portu jako wejścia. Funkcję rezystorów podciągajacych (ang. Pull-up) realizuje się za pomocą ustawienia na '1' odpowiednich bitów rejestru *portx*.

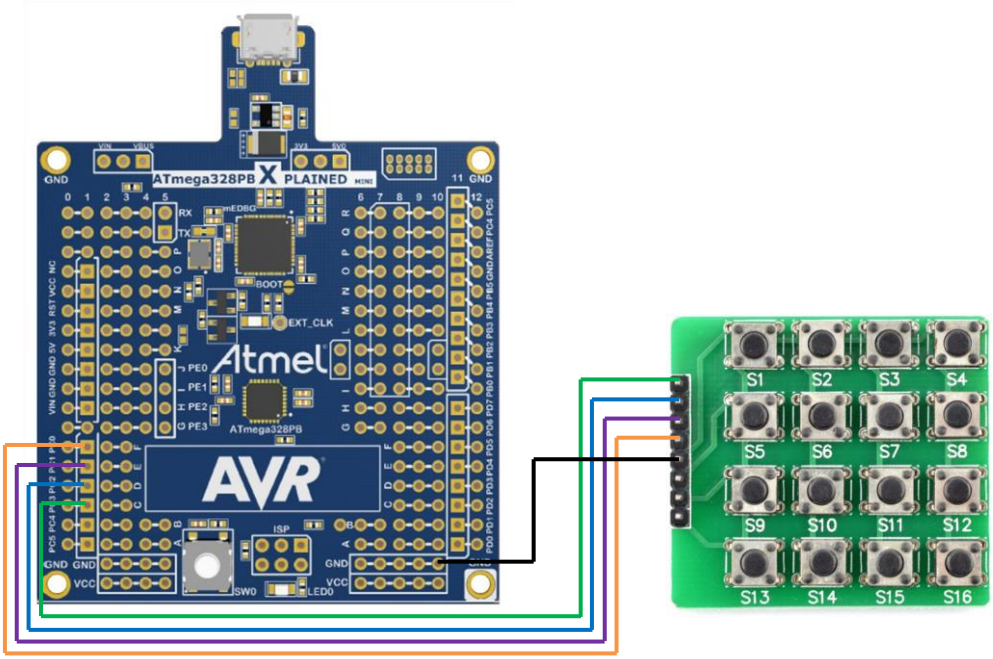
2.2. Sposób realizacji połączeń z modułem klawiatury

1. Podłącz moduł z diodami LED do portu B na płytce ATmega328PB Xplained mini zgodnie z opisem w instrukcji 1 laboratorium mikrokontrolerów (punkt 8.2).

2. Połącz sygnały z modułu klawiatury W1, W2, W3, oraz W4 do portu C (C0-C3) na płytce ATmega328PB Xplained mini (zgodnie z rysunkiem poniżej).

3. Połącz sygnał K1 z modułu klawiatury do masy (GND), aby ograniczyć funkcjonalność klawiatury do czterech przycisków (S1, S2, S3, S4).

4. Podłącz płytkę ATmega328PB Xplained mini do komputera.



3. Operacje na rejestrach wejścia/wyjścia

3.1. Porty wejścia/wyjścia

Mikrokontrolery megaAVR wykorzystywane na laboratorium oferują projektantowi cztery porty we/wy. Porty te są oznaczona jako: port B, port C, port D i port E. Przed użyciem porty te należy odpowiednio skonfigurować jako wejście lub wyjście. Każdy port posiada skojarzony z nim rejestr wejścia *pinX,* wyjścia *portX* oraz rejestr kierunku przesyłania danych *ddrX*. Poniższa tabela przedstawia adresy i opisy rejestrów dla portów B i C procesora AVR ATmega.

Register name	Memory space Address *(for sts, lds instructions)*	I/O address *(for in, out instructions)*	Usage
portb	$25	$05	output
ddrb	$24	$04	direction
pinb	$23	$03	input
portc	$28	$08	output
ddrc	$27	$07	direction
pinc	$26	$06	input

3.2. Rejestr DDRx

Porty mogą być użyte jako wejście lub wyjście. Rejestry *ddrX* używane są do ustawienia kierunku pracy danego portu. Aby ustawić dany port wyjściem należy zapisać wartość *0b11111111* do *ddra.* Aby ustawić port jako wejście należy wpisać zera do rejestru *ddrX.* Aby odczytać stan pinów wejścia czytamy rejestr *pinX. A*by ustawić końcówki portu skonfigurowanego jako wyjście nadpisujemy rejestr *portx.*

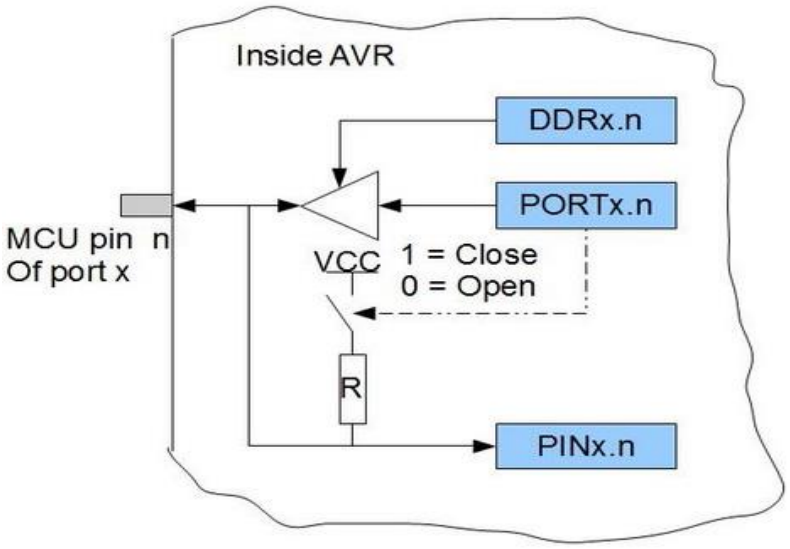


Diagram blokowy przedstawiony poniżej pokazuje schemat pojedynczej końcówki wejścia/wyjścia. Można zauważyć, iż rejestr *portX* może zostać użyty do skonfigurowania rezystora wewnętrznego procesora jako rezystora podciągającego. Aby użyć danego portu jako wejście, to w przypadku braku zewnętrznego rezystora podciągającego, rejestr *portX* należy wypełnić jedynkami. Aby odczytać dane z pinów portu wejścia/wyjścia czytamy rejestr *pinX,* natomiast aby zapisać dane na wyjście piszemy do rejestru *portX*.

3.3. Ładowanie danych z przestrzeni pamięci danych – instrukcja LDS

Instrukcja *lds* pozwala na przesłanie zawartości z przestrzeni adresowej pamięci danych do rejestru ogólnego przeznaczenia (r0-31). Źródłem przesyłania danych jest cała przestrzeń SRAM, czyli mogą to być rejestry wejścia/wyjścia, komórka pamięci SRAM lub rejestr ogólnego przeznaczenia (adresy od 0x00 do 0x1f).

```
lds     Rr,     K       ; load into register Rr the contents of the location K.
                        ; 0 ≤ r ≤ 31
                        ; k is an address between $0000 to $FFFF
```

Przykład:

```
lds r17, 0x25; load address $25 (port B) value into r17 register
```

3.4. Instrukcja odczytu rejestru wejścia/wyjścia

Instrukcja *in* pozwala na przesłanie zawartości rejestru wejścia/wyjścia do rejestru ogólnego przeznaczenia

```
in      Rr,     P       ;load the value of I/O location P to register Rr(r=0..31, P=0..63)
```

Uwaga

Instrukcja in odnosi się do rejestrów wejścia/wyjścia adresując je poprzez ich adres bezwzględny w przestrzeni wejścia/wyjścia. Przykładowo instrukcja 'in r20, 0x16' załaduje zawartość pod lokacją o adresie $16 w przestrzeni pamięci wejścia/wyjścia (adres ten widziany jest w całej pamięci danych jako $36) do rejestru r20.

3.5. In vs. lds

Jak wspomniano wcześniej, instrukcji *lds* można użyć do przesłania danych do rejestru ogólnego przeznaczenia z dowolnej lokacji w pamięci. Oznacza to, że możemy pobrać dane z rejestrów wejścia/wyjścia. Instrukcja *in* ma następujące zalety względem operacji *lds*:

1. *Lds* wykonuje się w dwóch taktach zegara, *in* zajmuje jeden takt.
2. *In* jest instrukcją dwu-bajtową, natomiast *lds* cztero-bajtową.
3. W przypadku *in* możemy używać nazw rejestrów wejścia/wyjścia zamiast adresów w pamięci.
4. Instrukcja *in* jest dostępna we wszystkich procesorach rodziny AVR, *lds* nie jest wspierana we wszystkich.

3.6. Czytanie z portu wejścia/wyjścia

Przykład:

Reading the value of ***port C***

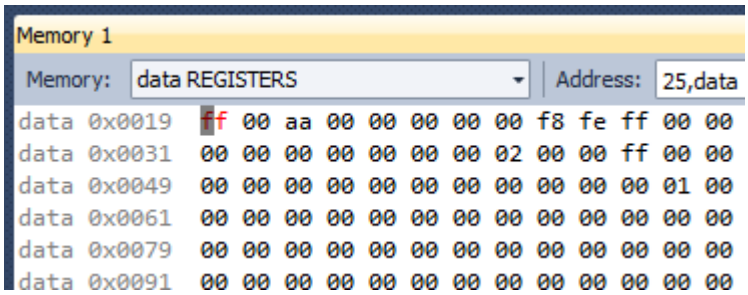
```
ldi r16, $ff      ; load immediate r16<= 0xFF
out $08, r16      ;writing ones(r16=0xFF) to portc value register to pull-up port registers
ldi r16, $00
out $07, r16      ;writing zeros to ddrc register to set input direction
in r16, $06       ;reading pinc value into r16 registers.
```

Ćwiczenie 2.1

1. Otwórz projekt w Microchip Studio. Można skorzystać z wcześniej skończonego projektu.
2. Napisz program, który czyta z portu C (przyciski) i wysyła tą wartość do portu B (ledy). Użyj instrukcji *rjmp* aby zakończyć program instrukcją skoku do samej siebie.
3. Przekrokuj program używając narzędzia AVR Debugger. Użyj opcji 'Memory:data REGISTERS', aby zobaczyć wartości rejestrów skojarzonych z portami wejścia/wyjścia w trakcie wykonywania programu.

Uwaga

Można wymusić zmianę zawartość pamięci danych w czasie, kiedy wykonywanie programu jest wstrzymane. W tym celu należy wpisać nowe wartości w oknie "Memory". Zmienione wartości będą wyświetlane w kolorze czerwonym.



4. Zaprogramuj procesor i uruchom programu.
5. Zachowaj program w pliku 'buttons_2_port.asm' w celu późniejszego wykorzystania w toku dalszych ćwiczeń.

4. Operacje bitowe

Asembler procesora AVR dostarcza instrukcji umożliwiających operacje na poszczególnych bitach rejestrów 8-bitowych. Instrukcje ta są szczególnie użyteczne, gdy w toku działania programu potrzebne jest ustawienia lub wyzerowanie pojedynczego bitu rejestru a wartość pozostałych bitów jest nieznana i ma pozostać niezmieniona.

4.1. Instrukcja ustawiania bitu w rejestrach wejścia/wyjścia (sbi)

Instrukcja *sbi* (*Set Bit in I/O register*) ustawia wyszczególniony bit w rejestrze wejścia/wyjścia. Instrukcja operuje na rejestrach wejścia/wyjścia (numeracja od 0 do 31).

sbi A, b ;Set bit b in an I/O register A $0 \leq A \leq 31, 0 \leq b \leq 7$

Example

Example:

sbi $0B, 7 ;Set bit 7 in Port D

4.2. Instrukcja zerowania bitu w rejestrach wejścia/wyjścia (cbi)

Instrukcja *cbi* (*Clear Bit in I/O register*) zeruje wyszczególniony bit w rejestrze wejścia/wyjścia. Instrukcja operuje na rejestrach wejścia/wyjścia (numeracja od 0 do 31).

cbi A, b ; Clear bit b in I/O register A $0 \leq A \leq 31, 0 \leq b \leq 7$

Example:

cbi $0B, 7 ; Clear bit 7 in Port D

Ponadto asembler AVR dostarcza instrukcje bitowe, które sprawdzają stany poszczególnych bitów rejestrów wejścia/wyjścia i w zależności od wartości danego bitu pomijają kolejną instrukcję programu. Są to instrukcje *sbis* i *sbic*.

4.3. Instrukcja Skip if Bit in I/O Register Cleared (sbic)

Instrukcja sprawdza czy dany bit jest wyzerowany, następnie w przypadku realizacji tego warunku nie wykonuje kolejnej instrukcji programu. Instrukcja operuje na rejestrach wejścia/wyjścia (numeracja od 0 do 31).

sbic A, b ; Test bit b in I/O register A 0 ≤ A ≤ 31, 0 ≤ b ≤ 7

Przykład:

```
wait:  sbic $08, 1     ;Skip next instruction if bit 1 in Port C is zero
       rjmp wait       ;
       nop             ;Continue (nop instruction means 'do nothing' – no operation)
```

4.4. Instrukcja Skip if Bit in I/O Register Set (sbis)

Instrukcja sprawdza czy dany bit jest ustawiony, następnie w przypadku spełnienia tego warunku nie wykonuje kolejnej instrukcji programu. Instrukcja operuje na rejestrach wejścia/wyjścia (numeracja od 0 do 31).

sbis A, b; Test bit b in I/O register A 0 ≤ A ≤ 31, 0 ≤ b ≤ 7

Przykład:

```
sbis $08, 2      ;Skip next instruction if bit 2 in Port C is set
sbi $05, 2       ;Set bit 2 in Port B
nop              ; Continue (nop - no operation)
```

4.5. Dyrektywa .INCLUDE

Dyrektywa *.include* pozwala na dodanie programu znajdującego się w innym podanym pliku. (podobnie jak dyrektywa #include w języku C). W celu użycia predefiniowanych nazw (np. nazw rejestrów) należy dołączyć dyrektywą *include* plik (*m328PBdef.inc*).

.include "m328PBdef.inc"

Przykład z użyciem nazw rejestrów:

```
1. sbis portc, 2        ; Skip next inst. if bit 2 in Port C is set
2. sbi portb, 5         ; Set bit 5 in Port B
3. out portc, r16       ;Copy r16 to portc
4. out ddrc, r16        ;Copy r16  to port C direction register
5. sbic portb, 1        ; Skip the next instruction if bit 1 in port B is zero
6. in r16, pinc         ;Reading port C value into r16 register.
```

Ćwiczenie 2.2

1. Przypisz przyciskowi S1 pojedynczą diodę LED. Napisz program, który testuje stan przycisku na porcie C i włącza lub wyłącza diodę w zależności od sprawdzanego stanu.
2. Użyj instrukcji *sbis/sbic*.
3. Przekrokuj program za pomocą narzędzia AVR Debugger.

4. Uruchom i przetestuj program na procesorze AVR.
5. Zachowaj program jako 'button_2_led.asm'.

5. Pętle w asemblerze procesora AVR

5.1. Rejestr statusowy AVR

Rejestr statusowy SREG to 8-bitowy rejestr potocznie zwany rejestrem flag. Poniższy rysunek prezentuje rysunek z wyszczególnionymi nazwami bitów tego rejestru.

Bit	7	6	5	4	3	2	1	0
SREG	I	T	H	S	V	N	Z	C

Bity C, Z, N, V, S oraz H to tzw. flagi warunkowe. Każda z nich może być wykorzystana do realizacji skoku warunkowego. W instrukcji zostanie przedstawione użycie flagi Z (flaga zero). Flaga ta oznacza, iż poprzednia instrukcja (ostatni, która modyfikuje tę flagę) dała wynik zero. Jeżeli wynik wykonania instrukcji jest zero wtedy flaga jest ustawiana (Z=1), w przeciwnym razie zerowana (Z=0).

5.2. Instrukcja Decement (dec)

Instrukcja odejmująca wartość jeden od zawartości rejestru Rd i zachowująca wynik operacji w rejestrze Rd, Rd: Rd<= Rd-1.

dec Rd; 0 ≤ d ≤ 31

5.3. Instrukcja skoku w przypadku nierówności (brne)

Instrukcja *brne* (*Branch if not Equal*) używa flagi zero w rejestrze statusowym. Instrukcja *brne* może być użyta w następujący sposób:

```
        ldi Rn, count   ; load Rn with the loop count
back:   ……..           ; Start of the loop
        ……..           ; Body of the loop
         dec Rn         ; Decrement Rn, Z=1 if Rn=0
        brne back       ; Branch to label back  if Z=0
```

Przykład

```
        ldi r17,$ff     ; Load a constant (loop count) in r17
delay:  nop
        nop
        dec r17          ; Decrement r17
        brne delay      ; Branch if r17<>0
```

5.4. Pętle zagnieżdżone

Jak zostało pokazane powyżej maksymalny licznik pętli może mieć wartość 255 (ograniczony jest rozmiarem rejestru). Aby zrealizować pętle o większej liczbie obiegów można zastosować pętle zagnieżdżone. W takim rozwiązaniu używamy dwóch rejestrów, aby przechowywać 16-bitowy licznik pętli.

Przykład:

```
                ldi r16,10          ;outer loop count 10
outter_loop:    ldi r17,20          ;inner loop count 20
inner_loop:
                nop                 ; any instructions within the loop
                dec r17
                brne inner_loop     ;repeat it 20 times
                dec r16
                brne outter_loop    ;repeat it 10 times
```

6. Inne instrukcje AVR

6.1. Instrukcja skoku w przypadku równości (breq)

Instrukcja *breq (Branch if Equal)* testuje flagę zero (Zero Flag (Z)) a następnie dokonuje rozgałęzienia (skok względny, jeżeli bit jest ustawiony).

breq k ;-64 ≤ k ≤ +63

6.2. Instrukcja Increment (inc)

Instrukcja dodająca wartość jeden do zawartości rejestru Rd i zachowująca wynik operacji w rejestrze Rd, Rd: Rd<= Rd+1.

inc Rd ;0 ≤ d ≤ 31

6.3. Instrukcja porównania (cp)

Instrukcja *cp (Compare)* wykonująca porównanie zawartości dwóch rejestrów (nie zmienia zawartości rejestrów). Jedynie wartość rejestru statusowego może ulec zmianie. Instrukcje warunkowe (*breq, brne*) mogą zostać użyte po tej instrukcji.

cp Rd, Rr ; 0 ≤ d ≤ 31, 0 ≤ r ≤ 31

Example

```
                ldi r16, $13        ;Initialize r16=$13
                ldi r17, $00        ;and r17=$00
loop:           nop                 ;do something useful
                inc  r17            ;increment r17 value
```

```
        cp r16, r17         ;and compare with r16
        brne loop           ;loop until r17 <$13 – repeat loop $13 times
stop:   rjmp stop
```

Ćwiczenie 2.3

1. Utwórz nowy plik 'mod_cnt.asm'
2. Napisz program, który inkrementuje wartość wyświetlaną za pomocą diod LED (wartości od 5 do 14). Użyj pętl zagnieżdżonych w celu realizacji opóźnień (diody powinny się zaświecać co 1 sekundę). Przyjmij jeden takt zegara na jedną instrukcję i częstotliwość taktowania 1MHz.
6. Przekrokuj program za pomocą narzędzia AVR Debugger.
7. Uruchom i przetestuj program na procesorze AVR.

7. Instrukcje skoków programowych

Skok względny (relative jump) to skok, dla którego adres skoku jest obliczany względem aktualnej wartości licznika programu (program counter) – PC<=PC+k. Skok krótki to taki skok, dla którego długość mniejsza jest niż 64 słowa (słowo=dwa bajty) - -64<=k<=+63. Instrukcje takie jak: rjmp, brne oraz breq to skoki krótkie o adresie względnym. Instrukcje ze skokiem krótkim to operacje 2-bajtowe. Adres podany ze skokiem krótkim to zawsze adres względny. Jeżeli adres ten jest większy od zera skok ten jest wykonywany wprzód w przeciwnym razie w tył.

7.1. Instrukcja długiego skoku (jmp)

Skok długi *jmp* to skok bezwarunkowy, którego adresem docelowym może być każda lokacja z przestrzeni adresowej pamięci procesora AVR (4M). Jest to 4-bajtowa instrukcja.

```
        jmp k           ;0 ≤ k < 4M
```

7.2. Instrukcja skoku względnego (rjmp)

Skok względny w przestrzeni adresowej z zakresu PC-2k+1 i PC+2k. Dla procesorów z pamięcią nie przekraczającą 4K słowa (8K bajtów) operacja ta może zaadresować całą pamięć z dowolnej lokacji programu.

```
        rjmp k          ;-2k ≤ k < 2k
```

Ćwiczenie 2.4

Czasami do realizacji programu potrzebna jest implementacja długich skoków warunkowych.

W celu ich realizacji wymagane jest połączenie warunkowych krótkich skoków ze skokami długimi. Kroki ćwiczenia:

1. Utwórz nowy plik 'mod_cnt_jmp.asm'.

2. Skopiuj i zmodyfikuj program 'mod_cnt.asm' w taki sposób, aby zrealizować długi skok warunkowy. Należy zastosować odpowiednią kombinację skoków: warunkowego krótkiego i bezwarunkowego długiego.