Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

Program Laboratorium
Rok akademicki 2003/2004. Semestr zimowy

Lp.	Seria	Temat ćwiczenia	Nr ćw.
1.	-	Zapoznanie z systemem DSM-51	-
2.	I	Linie wejść i wyjść mikrokontrolera	1
3.	I	Porty mikrokontrolera	2
4.	I	Pamięć wewnętrzna RAM. Organizacja i wykorzystanie stosu	3
5.	I	Operacje arytmetyczne	3B
6.	**-**	**Termin odróbkowy I serii ćwiczeń**	**-**
7.	II	Timery mikrokontrolera 8051. System przerwań	4
8.	II	System przerwań mikrokontrolera 8051	7
9.	II	Klawiatura przeglądana sekwencyjnie. Klawiatura matrycowa	9
10.	II	Wyświetlacz 7-segmentowy. Wyświetlacz alfanumeryczny LCD	10
11.	-	**Termin odróbkowy II serii ćwiczeń**	-
12.	III	Regulator tyrystorowy. Praca w czasie rzeczywistym	5
13.	III	Układy transmisji równoległej. Sterowanie światłami na skrzyżowaniu	6
14.	III	Przetwarzanie A/C i C/A. Model testera tranzystorów	8
15.	-	**Termin odróbkowy III serii ćwiczeń. Zaliczenie laboratorium.**	-

Spis treści

Literatura:

Piotr Gałka, Paweł Gałka, „Podstawy programowania mikrokontrolera 8051”, Zakład Nauczania Informatyki MIKOM, Warszawa 1995.

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ĆWICZENIE 1

LINIE WJEŚĆ I WYJŚĆ MIKROKONTROLERA

Celem ćwiczenia jest zapoznanie się ze sposobami sterowania portami mikrokontrolera 8051. Przedstawione zostanie wykorzystanie rozkazów adresujących pojedyncze linie w porcie oraz pełny port. W ćwiczeniu przedstawione zostaną metody sterowania urządzeniami podłączonymi bezpośrednio do portów mikrokontrolera.

Układ 8051 zawiera cztery 8-bitowe porty P0, P1, P2, P3. Wszystkie te porty mogą być adresowane jako całe bajty lub jako poszczególne bity. Wobec tego można korzystać z 32 linii wejścia / wyjścia. Możliwe jest to jednak tylko wówczas, gdy program jest wpisany do wnętrza mikrokontrolera. Jeżeli program umieszczony jest w zewnętrznej pamięci EPROM (tak jak w systemie DSM –51) lub wykorzystywana jest zewnętrzna szyna mikrokontrolera, to do bezpośredniego sterowania pozostaje tylko port P1 oraz 6 linii portu P3.

W strukturze wewnętrznej mikrokontrolera porty umieszczone są w obszarze rejestrów specjalnych (SFR – Special Function Register) [patrz dodatek X]. W każdy rejestr można wpisać 1 bajt informacji, czyli 8 bitów. Każdy z bitów jest w stanie 0 lub 1. w przypadku portów każdemu bitowi wpisanemu do rejestru portu odpowiada stan jednej linii. Każdy rejestr posiada swój adres, który służy do jego identyfikacji.

Port mikrokontrolera 8051 ma 8 linii, co odpowiada 8 końcówkom mikroprocesora, do których można podłączyć urządzenia zewnętrzne. Sterowanie urządzeń zewnętrznych odbywa się poprzez wpisanie odpowiednio na poszczególne bity stanu niskiego – 0 lub wysokiego – 1. Pamiętać należy, że po sygnale RESET wszystkie bity w portach są w stanie 1. Wpisany stan utrzymuje się aż do następnej operacji zapisu.

W ćwiczeniu wykorzystywane jest proste urządzenie zewnętrzne w postaci diody LED. Jest ona podłączona do linii 7 w porcie P1 i określana jest ona mianem diody świecącej TEST. Jeśli linia jest w stanie 0, to dioda świeci się, a gdy w stanie 1, to nie świeci się.

ZADANIE 1

„wprowadzanie programu przy użyciu komputera”
Należy odnaleźć plik l01_p1.asm znajdujący się w katalogu /lekcje/l01/. Po wykonaniu procesu asemblacji należy uruchomić ten program. Listing tego programu przedstawiony jest poniżej.

```
;Dioda TEST podłączona do linii 7 w porcie P1
;Linia ta oznaczona jest P1.7
;Stan 0 na linii zapala diodę

        LJMP START
        ORG 100H
START:

        CLR   P1.7    ;zeruj linię w porcie P1
                      ;czyli zapal diodę TEST
STOP:                 ;nie wykonuj innych działań
        LJMP  STOP    ;-pozostań w pętli STOP
```

WYJAŚNIENIE DZIAŁANIA PROGRAMU:

W celu zapalenia diody TEST należy wyzerować linię 7 w porcie P1. zerowanie pojedynczej linii można zrealizować poprze rozkaz CLR (clear – zeruj) za rozkazem należy podać adres bitu, który ma być zarezerwowany. W tym przypadku chodzi o bit 7 w porcie P1, oznaczony symbolem P1.7. Zawartość portu P1 po sygnale RESET wynosi 11111111_B. Wykonanie rozkazu:

CLR P1.7

Zmieni stan na linii 7 portu P1 na niski, co odpowiada świeceniu diody TEST. Po wykonaniu tego rozkazu mikroprocesor przechodzi do wykonania innych rozkazów. Należy pamiętać, że nie istnieje pojęcie „braku rozkazu", ponieważ mikroprocesor pobiera zawsze rozkazy z pamięci programu i realizuje je zgodnie z ich listą. Pamięć programu zawsze posiada pewną zawartość – jeżeli nie została ustawiona przez program, to jest ona losowa.. podstawową zasadą poprawnego programowania mikrokontrolerów jest zabezpieczenie się przed wykonywaniem losowej zawartości listy rozkazów.

Jeśli program wykonał już wszystkie zamierzone przez programistę czynności, to należy zatrzymać jego dalsza pracę, np. przez umieszczenie pustej pętli. W omawianym przykładzie jest to pętla STOP.

STOP: - jest to etykieta, która pozwala na odwołanie się do jej adresu. Adresu tego nie trzeba znać w czasie pisania programu – można posługiwać się etykietą. Pętlę realizuje rozkaz LJMP (LJMP – long jump – długi skok). Parametrem rozkazu jest adres miejsca w programie, do którego ma być wykonany skok. Przy określaniu tego adresu można użyć adresu. LJMP jest rozkazem skoku długiego, tj. może być wykonany do dowolnego adresu w pamięci programu.

OPRACOWANIE WYNIKÓW:

Jako rezultat wykonania zadania nr 1 należy:

1. Porównując przedstawiony powyżej listing programu z listingiem programu powstałego po procesie asemblacji podać, o jakie elementy został on uzupełniony oraz podać znaczenie tych uzupełnień
2. Narysować schemat algorytmu programu.

ZADANIE 2

„**wprowadzanie programu przy użyciu klawiatury systemu DSM-51"**

Przykład z zadania nr 1 wpisać przy pomocy klawiatury systemu DSM-51. W tym przypadku kod przyjmie postać:

```
CLR P1.7
L00:
LJMP L00
. END
```

Najistotniejsze różnice pojawiające się przy tworzeniu kodu źródłowego w wewnętrznym edytorze DSM-51 to:

- Brak możliwości wpisania komentarzy,
- Automatyczne umieszczanie programu pod adresem 100H,
- Możliwość użycia jedynie etykiet o nazwach: L00... L3F.

Po wykonaniu programu należy uruchomić go ponownie w trybie krokowym.

ZADANIE 3

„uruchamianie programu w trybie krokowym przy użyciu komputera"

Należy poddać asemblacji program o nazwie l01_p4. Następnie należy uruchomić go w trybie pracy krokowej. Należy zwrócić uwagę na informacje uzyskiwane na ekranie komputera w trakcie wykonywania kolejnych kroków. Listing programu przedstawiony jest poniżej:

```
LED EQU P1.7                    ;dioda TEST podłączona do P1.7

        LJMP  START
        ORG 100H
START:

LOOP:                           ;pętla mrugania diody TEST
        CLR   LED               ;zeruj linię – zapal diodę

        MOV   A,#10             ;czekaj czas 10*100ms  = 1s
        LCALL       DELAY_100MS ;podprogram z EPROMu

        SETB LED                ;ustaw linię – zgaś diodę

        MOV A,#10               ;czekaj czas 10*100ms  = 1s
        LCALL       DELAY_100MS

        LJMP  LOOP              ;powtórz
```

WYJAŚNIENIE DZIAŁANIA PROGRAMU:

W powyższym przykładzie dioda świecąca jest na przemian zapalana i gaszona. Oprócz poznanego już rozkazu CLR – zeruj bit istnieje rozkaz ustaw bit: SETB. Wykonywanie tych rozkazów na przemian spowoduje miganie diody TEST. Dzięki zastosowaniu nieskończonej pętli LOOP uniknięto wielokrotnego powtarzania ciągu rozkazów CLR i SETB. Należy zwrócić uwagę, że różnica pomiędzy zastosowaną w poprzednim przykładzie pętlą STOP a pętlą LOOP jest taka, że ta ostatnia powoduje wykonywanie rozkazów zawartych w jej wnętrzu (w pętli STOP nie było żadnego rozkazu).

Według algorytmu programu dioda świeci od momentu wykonania rozkazu CLR do momentu wykonania SETB. Natomiast od wykonania rozkazu SETB poprzez wykonanie rozkazu LJMP, aż do ponownego wykonania rozkazu CLR, dioda nie świeci.
W celu zapewnienia „obserwowalności" migotania diody program należy „ spowolnić". Pomiędzy rozkazy CLR i SETB zostały wstawione wywołania podprogramów realizujących opóźnienia czasowe. (podprogram ten stanowi wyposażenie systemu DSM-51). Podprogram ten powoduje „przeczekanie" przez mikroprocesor zadanego czasu A*100 ms, gdzie A jest zawartością akumulatora w momencie wywoływania podprogramu.
Akumulator jest rejestrem umieszczonym również w obszarze rejestrów specjalnych (patrz – opis systemu DSM-51). Podkreślić należy, że jest to podstawowy rejestr mikrokontrolera. Przed wywołaniem podprogramu DELAY_100MS (patrz listing), należy załadować do akumulatora odpowiednią wartość. Użyto do tego rozkazu:

MOV A,#10

który w ogólnej postaci wygląda tak:

MOV przeznaczenie, źródło

Rozkaz MOV(move – przesuń) powoduje przesunięcie bitu lub bajtu z miejsca określonego przez „źródło"(tj. #10) do miejsca określonego jako przeznaczenie (tj. A). Znaczek „#" określa, że chodzi bezpośrednio o wartość liczbową 10. Wobec tego wywołanie po tym rozkazie podprogramu DELAY_100MS spowoduje odczekanie 10*100 ms = 1s.

Wywołanie podprogramu (rozkazem LCALL) powoduje skok do podprogramu. Oznacza to, że następnym rozkazem po rozkazie LCALL będzie pierwszy rozkaz w danym podprogramie. Należy zapamiętać, że rozkaz LCALL różni się tym od rozkazu skoku (np. poznanego wcześniej rozkazu skoku LJMP) tym, że po zakończeniu podprogramu mikroprocesor potrafi powrócić do rozkazu umieszczonego po rozkazie LCALL. Szczegółowe omówienie tej tematyki będzie tematem jednego z kolejnych ćwiczeń.
Podsumowując, działanie omawianego programu można opisać następująco: po włączeniu diody świecącej TEST (rozkaz CLR LED) wykonywany jest skok do podprogramu DELAY_100MS, którego wykonanie trwa 1 s. Po tym czasie następuje powrót do programu głównego. Następuje wyłączenie diody (rozkaz SETB LED) i ponowny skok do podprogramu DELAY_100MS. Po zakończeniu tego podprogramu następuje powrót do programu i wykonanie instrukcji LJMP, która z kolei powoduje zamknięcie pętli i powrót do początku programu. Sekwencja włączenia / wyłączenia diody TEST będzie w ten sposób powtarzana.

OPRACOWANIE WYNIKÓW

Jako rezultat wykonania zadania nr 3 należy:

1. Podać, jakie informacje są dostępne w trakcie realizacji pracy krokowej przykładowego programu,
2. Wyjaśnić, jaki wpływ na program ma zastosowanie symbolu LED,
3. Narysować schemat algorytmu programu,
4. Jaki jest dostępny zakres opóźnień przy wykorzystaniu podprogramu DELAY_100MS

ZADANIE 4

„samodzielna analiza i modyfikacja przykładowego programu"

Należy poddać asemblacji program o nazwie l01_p6 a następnie go uruchomić. W programie oprócz urządzenia wyjścia w postaci diody LED, dodatkowo wykorzystuje się brzęczyk

(BUZZER), który podłączony jest do linii 5 portu P1. Operacje na bicie tej linii powodują włączanie i wyłączanie brzęczyka. Listing programu przedstawiony jest poniżej. Listing programu przedstawiony jest poniżej:

```
LED EQU P1.7                    ;dioda TEST podłączona do P1.7
BUZZER EQU      P1.5            ;brzęczyk podłączony do P1.5

        LJMP  START
        ORG 100H
START:

LOOP:                           ;pętla mrugania diody
                                ;i sterowania brzęczyka
        CPL   LED               ;zapal/zgaś diodę TEST
        CPL   BUZZER            ;włącz/wyłącz brzęczyk

        MOV   A,#10             ;czekaj czas 10*100ms  = 1s
        LCALL       DELAY_100MS ;podprogram z EPROMu

        LJMP  LOOP              ;powtórz
```

OPRACOWANIE WYNIKÓW:

Jako rezultat wykonania zadania nr 4 należy:

1. Na podstawie analizy działania przedstawionego wyżej programu określić wynik działania rozkazu CPL. Uzasadnić odpowiedź,
2. Narysować schemat algorytmu programu,
3. Zmodyfikować przykład z zadnia nr 4 tak, by świecenie diody TEST odbywało się na przemian z sygnałem z brzęczyka i jej stan zapalenia i wygaszania trwał dwukrotnie dłużej niż włączenie brzęczyka,
4. Narysować schemat algorytmu programu.

ĆWICZENIE 2

PORTY MIKROKONTROLERA

ZADANIE 5

„wpisywanie danych do portu”

Należy załadować plik l02_p1.asm znajdujący się w katalogu /lekcje/l02/. Po wykonaniu procesu asemblacji należy uruchomić ten program w trybie krokowym. Listing tego programu przedstawiony jest poniżej:

```
;Dioda TEST podłączona do linii 7 w porcie P1
LED_ON          EQU   01111111B

        LJMP  START
        ORG   100H
START:

        MOV   P1,#LED_ON ;wpisz 0 na bit 7 portu P1
                         ;wpisz 1-ki na bity 0..6
                         ;czyli zapal diodę TEST

        LJMP  $          ;pozostań w pętli
```

WYJAŚNIENIE DZIAŁANIA PROGRAMU:

W celu wpisania danych do portu, można posłużyć się rozkazem podanym w poprzednim ćwiczeniu – MOV. Rozkaz ten użyty był do załadowania stałej do akumulatora. W przedstawionym przykładzie wykorzystany on będzie do załadowania stałej do portu P1, tzn. do ustawienia poszczególnych linii portu w stan 1 lub 0.

Rozkaz MOV ładuje do portu P1 stałą LED_ON co powoduje włączenie diody świecącej TEST.

Załadowanie stałej do rejestru (w przykładzie wyżej – załadowanie stałej do rejestru portu) powoduje ustawienie poszczególnych bitów rejestru zgodnie z reprezentacją binarną wpisywanej liczby.

W przykładzie został użyty symbol „$”. Symbol ten oznacza aktualny adres, tzn. adres, pod którym w pamięci programu znajduje się dany rozkaz. Wobec tego rozkaz LJMP $ oznacza skok do adresu, w którym rozkaz ten się zaczyna. W rezultacie oznacza to nieskończone wykonanie tego rozkazu.

Wadą przedstawionego rozwiązania jest to, że rozkaz MOV ustawia jednocześnie stan wszystkich linii w porcie. Alternatywne rozwiązania, nieobciążone tą wadą przedstawione zostaną w kolejnych punktach.

OPRACOWANIE WYNIKÓW:

Jako rezultat wykonania zadania nr 1 należy:

1. podać jakie informacje są dostępne w trakcie realizacji pracy krokowej omawianego programu, omówić znaczenie tych danych dla zrozumienia pracy programu.

ZADANIE 6

„zerowanie i ustawianie linii portów przy pomocy logicznych operacji na parach bitów"

A) Należy załadować plik l02_p2.asm znajdujący się w katalogu /lekcje/l02/. Po wykonaniu procesu asemblacji należy uruchomić ten program w trybie krokowym. Listing tego programu przedstawiony jest poniżej:

```
;Dioda TEST podłączona do linii 7 w porcie P1
LED_ON          EQU   01111111B

        LJMP  START
        ORG   100H
START:

        ANL   P1,#LED_ON ;zeruj linię 7 portu P1
                         ;czyli zapal diodę TEST

        LJMP  $          ;pozostań w pętli
```

WYJAŚNIENIE DZIAŁANIA PROGRAMU:

Nowy rozkaz, jaki pojawił się w tym rozdziale, ANL, jest operacją wykonywaną na każdej parze bitów, niezależnie od pozostałych bitów i zgodnie z funkcją logiczną AND (tj. bit zerowy w rejestrze z bitem zerowym w stałej, bit pierwszy w rejestrze z bitem pierwszym w stałej. itd.).

B) Należy odnaleźć plik l02_p3.asm znajdujący się w katalogu /lekcje/l02/. Po wykonaniu procesu asemblacji należy uruchomić ten program w trybie krokowym. Listing tego programu przedstawiony jest poniżej:

```
;Dioda TEST podłączona do linii 7 w porcie P1
LED_ON          EQU   01111111B
LED_OFF         EQU   10000000B

        LJMP  START
        ORG   100H
START:

LOOP:                           ; pętla mrugania diody

        ANL   P1,#LED_ON        ;zeruje linię 7 portu P1
                                ;czyli zapal diodę TEST
        ORL   P1,#LED_OFF       ;ustawia linię 7 portu P1

                                ;czyli zgaś diodę TEST
                                itd.).
```

C) Należy odnaleźć plik l02_p4.asm znajdujący się w katalogu /lekcje/l02/. Po wykonaniu procesu asemblacji należy uruchomić ten program w trybie krokowym. Listing tego programu przedstawiony jest poniżej:

```
;Dioda TEST podłączona do linii 7 w porcie P1
LED_MASK  EQU   10000000B       ;maska do zmiany
                                ;stanu linii 7

        LJMP  START
        ORG   100H
START:

LOOP:                           ; pętla mrugania diody
        XRL   P1,#LED_MASK      ;neguj linię (0=>1, 1=>0)
                                ;zapal/zgaś diodę TEST

        MOV   A,#10             ;czekaj czas 10*100ms = 1s
        LCALL         DELAY_100MS   ;podprogram z EPROMu

         LJMP         LOOP      ;powtórz
```

WYJAŚNIENIE DZIAŁANIA PROGRAMU:

Nowy rozkaz, który pojawił się w tym rozdziale, XRL, jest operacją wykonywaną na każdej parze bitów, niezależnie od pozostałych bitów i zgodnie z funkcją logiczną XOR (tj. bit zerowy w rejestrze z bitem zerowym w stałej, bit pierwszy w rejestrze z bitem pierwszym w stałej. itd.).

OPRACOWANIE WYNIKÓW:

Jako rezultat wykonania zadania nr 2 należy:

1. Na podstawie poznanych do tej pory przykładów, przeanalizować wady i zalety przedstawionych w punktach A, B, C, rozkazów działających na poszczególnych bitach do sterowania pojedynczymi liniami portu jak i sterowania całym portem.
2. Opisać zmiany stanów podstawowych rejestrów mikrokontrolera w trakcie krokowego wykonywania przykładów A, B i C.

ZADANIE 7

„samodzielne opracowanie programu”

Jako rezultat wykonania zadania nr 3 należy:

1. Napisać i uruchomić program włączający i wyłączający jednocześnie diodę świecącą LED (linia 7 portu P1) i brzęczyk (linia 5 portu P1). Opisać stany rejestrów w trakcie pracy krokowej programu.
2. Zmodyfikować napisany program tak, aby świecenie diody LED odbywało się na przemian z włączeniem brzęczyka. Narysować schemat algorytmu programu.
3. Jakiego ciągu rozkazów należy użyć aby stan portu xx00x1x1 zmienić na stan xx10x0x1 (x – oznacza wartość nieznaną, która jednak nie może ulec zmianie w trakcie wykonywania programu).

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ĆWICZENIE 3

PAMIĘĆ WEWNĘTRZNA RAM. ORGANIZACJA I WYKORZYSTANIE STOSU

Wykonaj następujące zadania:

ZADANIE 1

Dokonaj operacji adresowania rejestrowego akumulatora:

```
        LJMP START
        ORG  100H
START:
        LCALL       LCD_CLR         ;wyczyść wyświetlacz LCD

        MOV A,#10H                  ;wpisz liczbę do A
        LCALL       WRITE_HEX       ;podprogram systemu DSM-51
                                    ;liczba z akumulatora
                                    ;na wyświetlaczu LCD

        MOV ACC,#20H                ;wpisz liczbę do ACC
        LCALL       WRITE_HEX       ;akumulator na LCD

        LJMP  $
```

Stosując pracę krokową przy użyciu komputera wpisz zawartość rejestru PSW oraz stan wyświetlacza LCD.

Bit	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0	LCD
Flaga	CY	AC	F0	RS1	RS0	OV	-	P	-

ZADANIE 2

Modyfikując program z przykładu 1 wykonaj adresowanie bezpośrednie wybranych bitów akumulatora. Wpisz zawartość rejestru PSW oraz stan wyświetlacza LCD.

Bit	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0	LCD
Flaga	CY	AC	F0	RS1	RS0	OV	-	P	-

ZADANIE 3

W jaki sposób rozróżnia się adres akumulatora od adresu poszczególnych bitów – do opracowania samodzielnego w sprawozdaniu.

ZADANIE 4

Wykonaj przykład 3.3. Stosując pracę krokową zaobserwować stan akumulatora, rejestru PSW oraz stan wyświetlacza LCD na przykładzie zawartości rejestru R7 w bankach 0,1,2,3

```
B0R7  EQU   7                   ;Rejestr R7 z banku 0
B1R7  EQU   8+7                 ;Rejestr R7 z banku 1
B2R7  EQU   10H+7               ;Rejestr R7 z banku 2
B3R7  EQU   18H+7               ;Rejestr R7 z banku 3

      LJMP START
      ORG  100H

START:

      MOV B0R7,#0               ;wpisz numer banku
      MOV B1R7,#1               ;do rejestru R7
      MOV B2R7,#2
      MOV B3R7,#3

      LCALL     LCD_CLR         ;wyczyść wyświetlacz LCD

                                ;bank 0
      MOV A,R7                  ;A<-R7=0
      LCALL     WRITE_HEX       ;akumulator na LCD

      SETB  RS0                 ;bank 1
      MOV A,R7                  ;A<-R7=1
      LCALL     WRITE_HEX

      SETB  RS1                 ;bank 3
      MOV A,R7                  ;A<-R7=3
      LCALL     WRITE_HEX

      CLR   RS0                 ;bank 2
      MOV A,R7                  ;A<-R7=2
      LCALL     WRITE_HEX

      LJMP$
```

ZADANIE 5

Modyfikując przykład 4 wykorzystując adresowanie pośrednie zapełnić wartością EE_H wybraną przestrzeń adresową. Podać listing programu – do opracowania samodzielnego.

ZADANIE 6

Stosują adresowanie bitowe ustaw zawartość komórki 21 na 3_H i wyświetlić ja na wyświetlaczu LCD. Podać listing programu.

ZADANIE 7

Modyfikując przykład 4 wykorzystując adresowanie indeksowe dokonaj zapełnienia obszaru pamięci zawartością AB_H – przeanalizować status wykorzystywanych zasobów (rejestrów) procesora - listing programu umieścić w sprawozdaniu.

ZADANIE 8

Wykonać przykład 5.1. Odczytaj zawartość akumulatora, rejestru PSW, wskaźnika stosu, zawartość komórek stosu.

```
        LJMP START
        ORG 100H
START:

        LCALL       LCD_CLR

        MOV  A,#'D'                 ;wpisz do A kod litery D
        PUSH ACC                    ;przechowaj akumulator na stosie
        LCALL       WRITE_DATA      ;wyświetl jako znak
                                    ;czyli litera D
        MOV  A,#'='                 ;wpisz znak równości
        LCALL       WRITE_DATA

        POP ACC                     ;pobierz wartość ze stosu
                                    ;do akumulatora
        LCALL       WRITE_HEX       ;wyświetl jako liczbę
                                    ;kod litery D = 44H
        SJMP  $
```

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ZADANIE 9

Wykonać przykład 5.2. Odczytaj zawartość akumulatora, rejestru PSW, wskaźnika stosu, zawartość komórek stosu.

```
        LJMP START
        ORG 100H
START:

        LCALL       LCD_CLR

        MOV  A,#137                     ;do A liczba 137
        ACALL       WRITE_BCD_HEX       ;wywołaj podprogram

        SJMP  $

;podprogram wpisuje liczbę z akumulatora
;najpierw dziesiętnie a następnie szesnastkowo
;podprogram nie zmienia zawartości rejestrów

WRITE_BCD_HEX
        PUSH PSW
        PUSH B                          ;przechowaj rejestry
        PUSH ACC

        ACALL       BIN_BCD             ;wywołaj podprogram
                                        ;zmiany liczby binarnej
                                        ;na liczbę BCD

        XCH  A,B                        ;zamień A <->B
                                        ;setki do A
                                        ;dziesiątki i jedn. do B
        LCALL       WRITE_HEX           ;wyświetl setki

        MOV  A,B                        ;dziesiątki i jedn. Do A
        LCALL       WRITE_HEX           ;wyświetl

        MOV  A,#'='
        LCALL       WRITE_DATA

        POP ACC                         ;odtwórz liczbę binarną
        PUSH ACC                        ;skopiuj ze stosu do A, czyli pobierz do A
                                        ;i poślij A z powrotem
        LCALL WRITE_HEX                 ;wyświetl

        MOV  A,#'H'
        LCALL       WRITE_DATA          ;dopisz H do liczby

        POP   ACC                       ;odtwórz rejestry
```

```
        POP   B
        POP   PSW

        RET                          ;powrót do podprogramu

;podprogram zamienia liczbę binarną z A
;na liczbę w kodzie upakowane BCD
;B – setki
;A – dziesiątki i jednostki

BIN_BCD
        MOV   B,#100       ;wydziel setki
        DIV   A,B          ;dzieląc przez 100
        PUSH  ACC          ;przechowaj setki

        MOV   A,B          ;wydziel dziesiątki
        MOV   B,#10        ;dzieląc przez 10
        DIV   AB

        SWAP  A            ;przesuń dziesiątki
        ORL   A,B          ;dodaj jednostki
        POP   B            ;odtwórz setki do B

        RET                ;koniec podprogramu
```

ZADANIE 10

Przeanalizuj ile miejsca na stosie potrzeba dla przykładu 5.2. Narysuj zawartość stosu w momencie schowania setek, w podprogramie BIN_BCD – do samodzielnego opracowania

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ĆWICZENIE 4.

TIMERY MIKROKONTROLERA 8051.

Wykonaj następujące zadania:

ZADANIE 1

„tryby pracy timerów mikrokontrolera”

Załadować do DSM-51 program **tryby.hex** a następnie uruchomić go w trybie „**Run**”. Przed wykonaniem kolejnego wariantu ustawień należy nacisnąć przycisk „**Reset RAM**”

Polecenia:

1a) ustawić

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*0*	*0*		-	*0*	*0*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	0010		*0000*	

Rejestr	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	-	*1*	-	*0*	-	-	-	-

Obserwować zwiększanie się stanu **timera 1**. Zauważyć, kiedy następuje zwiększanie **TH1**. Dla **16** kolejnych zmian **TH1** zanotować w tabeli po jednym stanie **timera 1** tuż przed każdą zmianą i po jednym stanie **timera 1** tuż po każdej zmianie. Jednocześnie zanotować stan 4 bitów rejestru **TCON**. Jaki wpływ na pracę **timera 1** ma wyzerowanie bitu **TR1** (klawisz 1)?

1b) ustawić:

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*0*	*0*		-	*0*	*0*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	FEFF		*0000*	

Rejestr	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	-	*1*	-	*0*	-	-	-	-

Obserwować zwiększanie się stanu **timera 1**. Zauważyć, kiedy następuje przepełnienie **timera 1**. Zanotować w tabeli po dwa stany **timera 1** tuż przed przepełnieniem i po dwa stany **timera 1** tuż po przepełnieniu. Jednocześnie zanotować stan 4 bitów rejestru **TCON**. Jaki wpływ na pracę **timera 1** ma wyzerowanie bitu **TR1** (klawisz 1)?

2a) ustawić:

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*0*	*1*		-	*0*	*0*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	0000		*0000*	

Rejestr	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	-	*1*	-	*0*	-	-	-	-

Zaobserwować, kiedy następuje zwiększanie **TH1**. Dla **3** kolejnych zmian **TH1** zanotować w tabeli po jednym stanie **timera 1** tuż przed każdą zmianą oraz po jednym stanie **timera 1** tuż po zmianie. Jednocześnie zanotować stan 4 bitów rejestru **TCON**. Jaki wpływ na pracę **timera 1** ma wyzerowanie bitu **TR1** (klawisz 1)?

2b) ustawić:

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*0*	*1*		-	*0*	*0*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	FE00		*0000*	

Rejestr	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	-	*1*	-	*0*	-	-	-	-

Zaobserwować, kiedy następuje przepełnienie **timera 1**. Zanotować w tabeli po dwa stany **timera 1** tuż przed przepełnieniem i po dwa stany **timera 1** tuż po przepełnieniu. Jednocześnie zanotować stan 4 bitów rejestru **TCON**. Jaki wpływ na pracę **timera 1** ma wyzerowanie bitu **TR1** (klawisz 1)?

3) ustawić:

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*0*	*2*		-	*0*	*0*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	55EE		*0000*	

Rejestr	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	-	*1*	-	*0*	-	-	-	-

(a) Zaobserwować, kiedy jest sygnalizowane przepełnienie **timera 1**. Zanotować dwa stany **timera 1** tuż przed przepełnieniem rejestru **TL1** oraz dwa stany **timera 1** tuż po przepełnieniu. Jednocześnie zanotować stan 4 bitów rejestru **TCON**. Jaki wpływ na pracę **timera 1** ma wyzerowanie bitu **TR1** (klawisz 1)?

Wykonać ponownie powyższe polecenia dla ustawień jak wyżej, za wyjątkiem stanu początkowego **timera 1**, przyjmując następujące ustawienia stanu początkowego: (b) 66EE, (c) 77EE, (d) 8800.

4) ustawić:

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*0*	*3*		-	*0*	*0*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	55EE		*0000*	

Rejestr	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	-	*1*	-	*0*	-	-	-	-

Zaobserwować pracę **timera 1**. Wynik obserwacji zanotować w tabeli. Jednocześnie zanotować stan 4 bitów rejestru **TCON**. Jaki wpływ na pracę **timera 1** ma wyzerowanie bitu **TR1** (klawisz 1)?

5) ustawić:

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*0*	*0*		-	*0*	*3*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	0000		*1199*	

Rejestr	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	-	*1*	-	*1*	-	-	-	-

(a) Zaobserwować i zanotować w tabeli dwa stany **TL0** przed ustawieniem bitu **TF0** oraz dwa stany **TL0** po ustawieniem bitu **TF0**. Zaobserwować i zanotować w tabeli dwa stany **TH0** przed ustawieniem bitu **TF1** oraz dwa stany **TH0** po ustawieniem bitu **TF1**. Jak zachowuje się **timer 1**? Jaki wpływ na pracę **timera 0** ma wyzerowanie bitu **TR1** (klawisz 1) a jaki wpływ ma wyzerowanie bitu **TR0** (klawisz 0)?
Wykonać ponownie powyższe polecenia dla ustawień jak wyżej, za wyjątkiem stanu początkowego **timera 0**, przyjmując następujące ustawienia stanu początkowego: (b) 9911, (c) 7777.

5d) ustawić:

Rejestr	Gate	C/~T	M1	M0	Gate	C/~T	M1	M0
TMOD	-	*1*	*0*		-	*0*	*3*	

Stany począt-kowe:	TIMER 1		TIMER 0	
	TH1	TL1	TH0	TL0
	FFEF		*1199*	

Rejestr	**TF1**	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**
TCON	-	***1***	-	***1***	-	-	-	-

Zwiększać stan **timera 1** posługując się klawiaturą sekwencyjną. Zaobserwować czy przepełnienie **timera 1** ma wpływ na **TF1**? Wynik obserwacji umieścić w tabeli.
6) Wykonać polecenia 1b), 2b) i 3a) ustawiając w rejestrze **TCON** dla **timera 1** bit **C/~T=1**. Stan **timera 1** zwiększać posługując się klawiaturą sekwencyjną. Zwrócić uwagę na możliwość powstawania szeregu impulsów w chwili puszczenia klawisza (odbicia).
10) Na zakończenie Zadania 1 i Zadania 2 opracować tabelę opisującą działanie **timera 1** i **timera 2** w zależności od trybu pracy i stanów **TR1** i **TR0**.

Bity sterujące				
C/T	**M1**	**M0**	**TRx**	**Opis działania**

Wzór tabeli:

		G	C/T	M1	M0	
Tryb pracy :						
TH	TL	TF1	TR1	TF0	TR0	UWAGI

ZADANIE 2

„zliczanie impulsów zewnętrznych timerze mikrokontrolera”

Zadaniem programu **licznik.asm** jest włączanie i wyłączanie LED co **N** - określoną ilość impulsów z klawiatury sekwencyjnej zliczanych w jednym z timerów. Liczbę impulsów **N** - przed wykonaniem zadania określi prowadzący zajęcia.
W treści programu uzupełnić miejsca zaznaczone znakami zapytania **licznik.asm** tak, aby wykonywał on wyżej opisane funkcje. Dokonać asemblacji programu i uruchomić na DSM-51

```
LED    EQU   P1.7

;********* Ustawienie TIMERów *********
TMOD_SET  EQU   ???
TH?_SET           EQU   ???
TL?_SET           EQU   ???
;*************************************

        LJMP  START
        ORG   100H
START:
        LCALL         INICJALIZACJA
        ???     ???                    ;ustaw tryb pracy timerów
```

```
        ???     ???                     ;ustaw stan początkowy
        ???     ???                     ;wybranego timera

        ???     ???                     ;start wybranego timera
        CLR     LED                     ;włącz LED
LOOP:
        LCALL           WYSWIETL_STANY_LICZNIKOW
        ???     ???,LOOP                ;pętla dopóki nie przepełniony
        ???     ???                     ;zeruj flagę
        CPL     LED                     ;przełącz LED
        SJMP    LOOP

INICJALIZACJA:
        ..............................
        RET

WYSWIETL_STANY_LICZNIKOW:
        ..............................
        RET
```

ZADANIE 3

„odliczanie czasu w timerze mikrokontrolera”

Zadaniem programu **czas.asm** jest włączanie i wyłączanie LED co **T** - określony odcinek czasu. Odmierzanie czasu wykonywane jest za pomocą **85H** - krotnego zliczania impulsów zegara systemowego (1,085µs) w wybranym timerze. Długość odcinka czasu **T** - przed wykonaniem ćwiczenia określi prowadzący zajęcia. Uzupełnić treść programu **czas.asm** tak, aby wykonywał wyżej opisane funkcje. Dokonać asemblacji programu i uruchomić na DSM-51

```
LED     EQU     P1.7

;********* Ustawienie TIMERów *********
TMOD_SET        EQU     ???
TH?_SET         EQU     ???     ;stany początkowe
TL?_SET         EQU     ???
KROTNOSC        EQU     85H
;*************************************

        LJMP    START
        ORG     100H
START:
        ???     ???                     ;tryb pracy timerów

        ???     ???                     ;stan początkowy
        ???     ???                     ;wybranego timera
```

```
        ???     ???                 ;start wybranego timera
LOOP:                               ;pętla mrugania diody TEST
        CPL     LED
        MOV     A,#KROTNOSC         ;odczekaj czas KROTNOŚĆ*???ms=???s
TIME:
        ???     ???,$               ;czekaj, aż wybrany timer
                                    ;odliczy ???ms
        ???     ???                 ;wybrany timer na ???ms
        ???     ???
        ???     ???                 ;zerowanie flagi
                                    ;wybranego timera
        DJNZ    ACC,TIME            ;odczekanie N*???ms
        SJMP    LOOP
```

ZADANIE 4

„przerwania, odliczanie czasu w timerze mikrokontrolera”

Uruchom na DSM-51 program **LED.hex.** Zadaniem programu jest włączanie i wyłączanie LED co okres odmierzany w timerze przy zastosowaniu przerwań.

Przeanalizować, a następnie skomentować listing programu.

ZADANIE 5

„przerwania, odliczanie czasu w timerze mikrokontrolera”

Uruchom na DSM-51 program **przerwan.hex.** Zadaniem programu jest pobieranie znaku 0..9 z klawiatury i wysyłanie na wyświetlacz oraz jednoczesne (w tle) włączanie i wyłączanie LED co okres odmierzany w timerze.

Sprawdzić, czy program prawidłowo wyprowadza znaki na wyświetlacz. W jaki sposób może tu dochodzić do wpływania na siebie programu i procedury obsługi programu. Przeanalizować listing programu pod tym kątem. Zaproponować wykorzystanie stosu do usunięcia występujących trudności. Zmodyfikować, zasemblować i uruchomić program.

```
LED     EQU     P1.7

;********* Ustawienie TIMERÓW *********
;TIMER 0
T0_G    EQU     0                       ;GATE
T0_C    EQU     0                       ;COUNTER/-TIMER
T0_M    EQU     1                       ;MODE (0..3)
TIM0    EQU     T0_M+T0_C*4+T0_G*8      ;TIMER 1

T1_G    EQU     0                       ;GATE
T1_C    EQU     0                       ;COUNTER/-TIMER
T1_M    EQU     0                       ;MODE (0..3)
```

```
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

TH0_SET         EQU   0EAH        ;stan początkowy Timera 0
TL0_SET         EQU   066H        ;opóźnienia 6ms

KROTNOSC  EQU   0FFH              ;krotność powtarzania opóźnień 6ms

;*************************************
        LJMP  START
;******** Przerwanie Timer 0 ********
        ORG   0BH

        MOV   TH0,#TH0_SET        ;stan początkowy do TH0
        MOV   TL0,#TL0_SET        ;i do TL0 (6ms)
        DJNZ  B,JESZCZE_NIE       ;czy wykonano KROTNOSC opóźnień,
                                  ;tj. KROTNOSC*6ms?
        CPL   LED                 ;mruganie diody TEST
        MOV   B,#KROTNOSC         ;odczekaj kolejną KROTNOSC opóźnień
                                  ;KROTNOSC*6ms
JESZCZE_NIE:
        RETI
;*************************************
        ORG   100H
START:
        MOV   TMOD,#TMOD_SET      ;Timer 0 liczy czas
        MOV   TH0,#TH0_SET        ;stan początkowy do TH0
        MOV   TL0,#TL0_SET        ;i do TL0
        SETB  TR0                 ;start Timera 0

        SETB  EA                  ;włącz zezwolenie ogólne na przerwania
        SETB  ET0                 ;włącz zezwolenie na przerwanie od Timera 0
;*****************************************************************
;
;       Robocza część programu głównego
;
;*****************************************************************
LOOP_0:
        LCALL     LCD_CLR         ;czyść wyświetlacz
        MOV       R1,#16          ;do odlicz. max. 16 znaków na wyświetl.

LOOP:                             ;w tej pętli czyta/pisze znak
        LCALL     WAIT_KEY        ;czyta klawisz
        MOV       B,A             ;przechowuje znak

        MOV       A,#2            ;niewielkie
        LCALL     DELAY_MS        ;opóźnienie
```

```
MOV  A,B          ;pobiera znak z "przechowalni"
ADD  A,#30H       ;zamienia binarny kod cyfry na kod w ASCII

MOV  R0,#LCDWD    ;wysyła
MOVX     @R0,A    ;znak na wyświetlacz

DJNZ R1,LOOP      ;wyczyść wyświetlacz, jeżeli było już 16 znaków
SJMP LOOP_0       ;zamknij pętlę
```

DODATEK

Program **tryby.hex** umożliwia obserwację stanów rejestrów zliczających timerów, oraz stanów 4-ech najstarszych bitów rejestru **TCON**.

Timer 1:

st. 2 cyfry hex.	*m³. 2 cyfry hex.*
TH1 (8 bitów)	*TL1 (8 bitów)*

Timer 0:

st. 2 cyfry hex.	*m³. 2 cyfry hex.*
TH0 (8 bitów)	*TL0 (8 bitów)*

Rejestr TCON:

TF1	*TR1*	*TF0*	*TR0*	-	-	-	-

W celu umożliwienia obserwacji pracy timerów we wszystkich przewidzianych warunkach pracy program żąda wprowadzenia:
- trybu pracy (ustawienia w rejestrze **TMOD**):

Timer 1				*Timer 0*			
-	*C/~T*	*M1*	*M0*	-	*C/~T*	*M1*	*M0*

C/~T - *wybór źródła sygnału zliczanego*
M1 - *tryb pracy*
M0 - *tryb pracy*

- stanów początkowych rejestrów zliczających timerów:
- początkowych stanów bitów **TR1** i **TR0**.

Powyższe dane wprowadza się następująco:

```
Timer/liczn? 0/1
T1/T0: x/y
```

gdzie: ***x*** - bit C/T **Timera 1**, ***y*** - bit C/T **Timera 0**; ***x,y***=0 - timer, ***x,y***=1 - licznik.

```
Tryby pracy     :
T1/T0: w/z
```

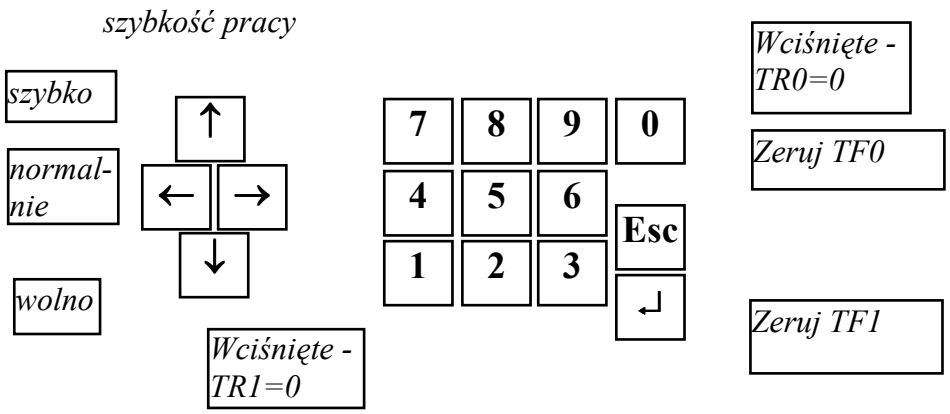
gdzie: ***w*** - tryb pracy **Timera 1**, ***z*** - tryb pracy **Timera 0**; ***w,z***∈<0,3>

```
Stany początk.  :
T1/T0: aaaa/bbbb
```

gdzie: ***aaaa*** - stan pocz. **Timera 1**, ***bbbb*** - stan pocz. **Timera 0**; ***aaaa,bbbb***∈<0000,FFFF>

```
Stany TR1/TR0   :
T1/T0: p/q
```

gdzie: ***p*** - bit TR1, ***q*** - TR0; ***p,q***∈<0,1>
Poniżej opisano dodatkowe funkcje realizowane przy użyciu klawiatury.


szybkość pracy
szybko
normal-
nie
wolno
↑
←
→
↓
Wciśnięte -
TR1=0
7
8
9
0
4
5
6
1
2
3
Esc
↵
Wciśnięte -
TR0=0
Zeruj TF0
Zeruj TF1

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ĆWICZENIE 5

REGULATOR TYRYSTOROWY. PRACA W CZASIE RZECZYWISTYM

Wymagania

Przed przystąpieniem do ćwiczenia należy znać:
a) typy adresowania: bitowego i indeksowego,
b) skoki warunkowe,
c) działanie i programowanie timerów,
d) odczyt stanu linii wejściowej.

ZADANIE 1

„sterowanie tyrystorem”

Załadować do DSM-51 program **żarówka.hex** a następnie uruchomić go w trybie „**Monitor**”. Program steruje jasnością świecenia żarówki poprzez włączanie płynącego przez nią prądu z odpowiednim opóźnieniem względem momentu przejścia napięcia sieci przez '0' w każdej połówce okresu tego napięcia. Jest to tzw. sterowanie fazowe. Program czyta klawiaturę matrycową (klawisze **1..7**) i zgodnie z przyciśniętym klawiszem ustawia opóźnienie odpowiednio **1..8**ms. Wartość opóźnienia jest wypisana na wyświetlaczu LCD.

Prześledzić krokowo (klawisz „**spacja**”), z pominięciem procedur (klawisz „**N**”), wykonanie programu. Zaobserwować i wpisać do poniższej tabeli stany występujące w kolejnych krokach (w tabeli „**K**”) **po wykonaniu danej instrukcji**:
a) akumulatora (w tabeli „**A**”),
b) portu P1 (w tabeli „**P1**”),
c) żarówki (w tabeli „**Ż**”),
d) wyświetlacza (w tabeli „**W**”).
Uzupełnić brakujące komentarze do programu źródłowego.

K	Program źródłowy	A	P1	Ż	W	Komentarz
1	START: LCALL INICJALIZACJA					;ustawienia wstępne
2	LCALL WYSWIETL					;wyświetla wstępną wart opóźn. (opóźn. w A)
3	LOOP: JNB SIEC,$					;oczekiwanie na "0" sieci
...	LCALL DELAY_MS					;odczekanie opóźnienia A*1ms
...	CLR BRAMKA					;*???????*
... ...	MOV A,#10					;*???????*
...	DJNZ ACC,$					;*???????*
...	SETB BRAMKA					;*???????*
...	LCALL KLAWIATURA					;pobiera do A nową wartość opóźnienia
...	SJMP LOOP					;zamknięcie pętli programu

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

Uwaga:

SIEC	EQU	**P3.4**	;wejście z transoptora	**0** - jest napięcie na żarówce **1** - "0" sieci
BRAMKA	EQU	**P1.2**	;bramka tyrystora	**0** - włączenie tyrystora **1** - niewłączanie tyrystora

ZADANIE 2

„adresowanie indeksowe."

Załadować do DSM-51 program **tabela.hex** a następnie uruchomić go w trybie „**Monitor**". Program pobiera argument funkcji (x) z klawiatury (klawisze 1..7) a następnie odczytuje odpowiadającą jej wartość funkcji. Wartością funkcji jest liczba dwubajtowa podawana w R7 (starszy bajt) i w R6 (młodszy bajt). Wartość argumentu i wartość funkcji wypisywane są na wyświetlaczu.

Dla wybranych kolejno trzech wartości argumentu funkcji prześledzić krokowo (klawisz „**spacja**"), z pominięciem procedur (klawisz „**N**"), wykonanie programu. Klawisz wybierający wartość argumentu funkcji przyciskać w trakcie wywoływania procedury KLAWIATURA (**LCALL 014FH**). Zaobserwować i wpisać do poniższej tabeli stany występujące w kolejnych krokach **po wykonaniu danej instrukcji**:

a) rejestru **DPTR** (w tabeli „**D**"),
b) akumulatora (w tabeli „**A**"),
c) rejestru **R7**,
d) rejestru **R6**,
e) klawiszy (nr wciśniętego klawisza - „**KL**"),
f) wyświetlacza (w tabeli „**LCD**").

Przepisz zawartość tabeli funkcyjnej.

Program źródłowy	D	A	D+A	R7	R6	KL	LCD	Komentarz
START: LCALL INICJALIZACJA								;ustawienia wstępne
LOOP: LCALL KLAWIATURA								;czyta klawiaturę
MOV DPTR,#TABELA								;tabela funkcyjna
PUSH ACC								;przech. nr klawisza
RL A								;mnóż przez 2
PUSH ACC								;przech przesunięcie
MOVC A,@A+DPTR								;starszy bajt z tabeli
MOV R7,A								;wynik do R7
POP ACC INC A MOVC A,@A+DPTR								;odtwórz przesun. ;wskaż młodszy bajt ;pobierz mł bajt
MOV R6,A								;wynik do R6
POP ACC								;odtwórz nr klaw.
SJMP LOOP								;pętla programu

Tabela funkcyjna

	0	1	2	3	4	5	6	7
0180								
0188								

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ZADANIE 3

„sterowanie fazowe, odliczanie czasu w timerze mikrokontrolera”

Uzupełnić program **timer.asm** tak, aby opóźnienie czasu określane z klawiatury odmierzane było w **Timerze 0** mikrokontrolera. Obliczyć wartości stanów początkowych timera dla poszczególnych opóźnień (klawisz 1 - 1ms, klawisz 2 - 2ms, itd.) i umieścić je w tabeli funkcyjnej. Tak uzupełniony program źródłowy zasemblować i uruchomić.

```
B0R2  EQU  2              ;adres rejestru 2 w banku 0
B0R6  EQU  6
B0R7  EQU  7
SIEC        EQU  P3.4     ;wejście z transpotora
                          ;0 - jest napięcie na żarówce
                          ;1 - "0" sieci
BRAMKA      EQU  P1.2     ;wyjście przez transoptor - bramka tyrystora
                          ;0 - włączenie tyrystora
                          ;1 - niewłączanie tyrystora

;********* Ustawienie TIMERÓW *********
;TIMER 0
T0_G  EQU  ???     ;GATE
T0_C  EQU  ???     ;COUNTER/-TIMER
T0_M  EQU  ???     ;MODE (0..3)
TIM0  EQU  T0_M+T0_C*4+T0_G*8
;TIMER 1
T1_G  EQU  0       ;GATE
T1_C  EQU  0       ;COUNTER/-TIMER
T1_M  EQU  0       ;MODE (0..3)
TIM1  EQU  T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU  TIM0+TIM1*16
;*************************************
        LJMP  START
;*************************************
        ORG  0100H
START:
        LCALL       INICJALIZACJA     ;ustawienia początkowe (R7 i R6)
        ???    ???,???       ;???
LOOP:
        ???    ???,???       ;???
        ???    ???,???       ;???

        ???    ???,???       ;???
        ???    ???,???       ;???
        ???    ???,???       ;???

        ???    ???,???       ;???
        ???    ???,???       ;???
```

```
        ???     ???,???         ;???

        CLR     BRAMKA                  ;impuls włączający tyrystor
        MOV     A,#10
        DJNZ    ACC,$
        SETB    BRAMKA

        LCALL           KLAWIATURA      ;sprawdza/czyta nowa wart opóźnienia, numer
klawisza do A
        SJMP    LOOP                    ;stan początkowy timera do R7 i R6
;*********************************
;       Procedury
;*********************************
INICJALIZACJA:                          ;ustawienia początkowe
        ...............................
        RET                             ;koniec procedury

KLAWIATURA:                             ;pobiera nowa wartość opóźnienia
        ...............................
        RET                             ;koniec procedury

TABELA:
        DB      0FFH,0FFH
        DB      0FFH,0FFH
        DB      0FFH,0FFH
        DB      0FFH,0FFH
        DB      0FFH,0FFH
        DB      0FFH,0FFH
        DB      0FFH,0FFH
        DB      0FFH,0FFH
```

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ZADANIE 4

„pomiar częstotliwości sieci przy użyciu timerów”

Uzupełnij program źródłowy **sieć.asm** tak, aby mierzył on częstotliwość sieci. Przed napisaniem programu wykonaj schemat blokowy. Program źródłowy poddaj asemblacji i uruchom.

Uwaga:
Procedura WYŚWIETL wypisuje na wyświetlaczu wartość częstotliwości podaną procedurze w akumulatorze.

```
SIEC   EQU   P3.4                    ;wejście z transpotora
                                     ;0 - jest napięcie na żarówce
                                     ;1 - "0" sieci
BRAMKA     EQU   P1.2                ;wyjście przez transoptor - bramka tyrystora
                                     ;0 - załączenie tyrystora
                                     ;1 - niezałączanie tyrystora
;******** Ustawienie TIMERÓW********
;TIMER 0
T0_G  EQU   0                        ;GATE
T0_C  EQU   ???                      ;COUNTER/-TIMER
T0_M  EQU   ???                      ;MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8       ;TIMER 1
T1_G  EQU   0                        ;GATE
T1_C  EQU   ???                      ;COUNTER/-TIMER
T1_M  EQU   ???                      ;MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

TH1_SET           EQU   ???
TL1_SET           EQU   ???
;*************************************
        LJMP  START
;*************************************
        ORG   0100H
START:
        MOV   TMOD,???
LOOP:
        ????????
        ........
        ????????

        MOV   A,TL0
        MOV   B,#2
        DIV   AB
        LCALL         WYSWIETL
        CPL   P1.7
        SJMP  LOOP
```

```
WYSWIETL:
        ..........
        RET
```

ZADANIE 5

„płynne włączanie i wyłączanie świateł”

Uzupełnij program źródłowy **kino.asm** tak, aby - zależnie od polecenia z klawiatury - płynnie włączał i wyłączał żarówkę. Czas płynnego włączania i wyłączania powinien wynosić **5** sekund.

Uwaga:

Procedura **KLAWIATURA** sprawdza stan klawiszy. Jeżeli przyciśnięto klawisz ↑ procedura kończy się ustawieniem bitu **F0** (w rejestrze PSW) i ustawieniem bitu **PSW.1** (komenda „**ZAPALAJ**”). Jeżeli przyciśnięto klawisz ↓ procedura kończy się ustawieniem bitu **F0** (w rejestrze PSW) i wyzerowaniem bitu **PSW.1** (komenda „**ZGAŚ**”). Jeżeli nie przyciśnięto żadnego z w/w klawiszy procedura nie zmienia stanów w/w bitów.

Do zmian opóźnienia fazowego zaproponowano procedury: **SKRACAJ** i **WYDLUZAJ**. W wyniku działania każdej z procedur nowe wartości stanów początkowych timera wpisywane są do rejestrów R7 (do wpisania do TH) i R6 (do wpisania do TL).

Procedura **SKRACAJ**, jeżeli maksymalnie skrócono opóźnienie fazowe kończy się wyzerowaniem bitu **F0** (w rejestrze PSW) i ustawieniem bitu **PSW.1** (stan żarówki „**ZAPALONE**”). Jeżeli nie skrócono maksymalnie opóźnienia fazowego procedura ta kończy się bez zmiany w/w bitów.

Procedura **WYDLUZAJ**, jeżeli maksymalnie wydłużono opóźnienie fazowe kończy się wyzerowaniem bitu **F0** (w rejestrze PSW) i wyzerowaniem bitu **PSW.1** (stan żarówki „**ZGASZONE**”). Jeżeli nie wydłużono maksymalnie opóźnienia fazowego procedura ta kończy się bez zmiany w/w bitów.

```
;Kombinacje stanów flag: F0 i PSW.1 mają następujące znaczenia:
;
;               F0      PSW.1 stan automatu
;               0       0       "ZGASZONE"
;               0       1       "ZAPALONE"
;               1       0       "GASZENIE"
;               1       1       "ZAPALANIE"
;*************************************
SIEC    EQU     P3.4                    ;wejście z transpotora
                                        ;0 - jest napięcie na żarówce
                                        ;1 - "0" sieci
BRAMKA  EQU     P1.2                    ;wyjście przez transoptor - bramka tyrystora
                                        ;0 - załączenie tyrystora
                                        ;1 - niezałączanie tyrystora

MIN_TH  EQU     0E0H                    ;min. wart. stanu pocz. Timera
MIN_TL  EQU     000H

MAX_TH  EQU     0FFH                    ;max. wart. stanu pocz. Timera
```

```
MAX_TL      EQU   0FFH

DELTA       EQU   ???H                  ;przyrost stanu pocz. przypad. na każdy okres

F_CSKB1     EQU   0FF22H                ;adres bufora klawiatury; klawisze 8..F
;********* Ustawienie TIMERÓW*********
;TIMER 0
T0_G  EQU   0                           ;GATE
T0_C  EQU   0                           ;COUNTER/-TIMER
T0_M  EQU   1                           ;MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8          ;TIMER 1
T1_G  EQU   0                           ;GATE
T1_C  EQU   0                           ;COUNTER/-TIMER
T1_M  EQU   0                           ;MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16
;************************************
;************************************
        LJMP  START

;************************************
        ORG   0100H
START:
        LCALL       LCD_CLR

        ORL   TMOD,#TMOD_SET

        MOV   R7,#MIN_TH
        MOV   R6,#MIN_TL
        CLR   F0                        ;stan "ZGASZONE"
        CLR   PSW.1                     ;sygnalizuje podając F0=0 i PSW.1=0
LOOP:
        LCALL       KLAWIATURA
        ???   ???                       ;???
        ???   ???                       ;???

ZAPAL:
        LCALL SKRACAJ
        SJMP  OBSLUZ_TYRYSTOR
ZGAS:
        LCALL       WYDLUZAJ

OBSLUZ_TYRYSTOR:
        JNB   SIEC,$                    ;czeka na przejście napięcia przez 0

        ???   ???                       ;stany początkowe z R7 R6 do TH0 TL0
        ???   ???
        ???   ???                       ;uruchom Timer 0
```

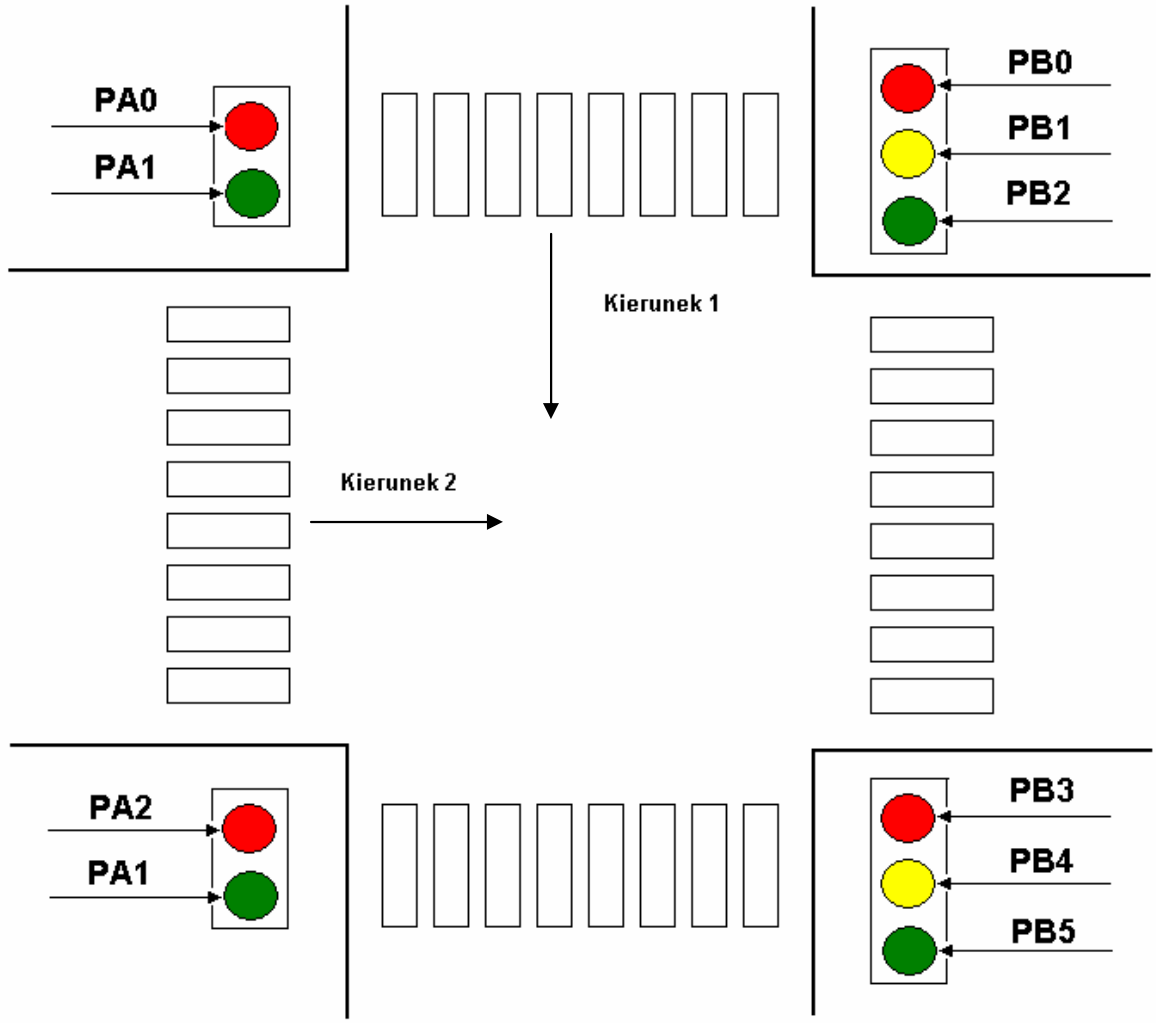
```
        ???     ???                     ;czeka aż Timer 0 przepełni się
        ???     ???                     ;zatrzymaj Timer 0
        ???     ???

        CLR     BRAMKA                  ;impuls załączający tyrystor
        MOV     A,#10
        DJNZ    ACC,$
        SETB    BRAMKA

        SJMP    LOOP                    ;zamknij pętlę programu


KLAWIATURA:
        MOV     DPTR,#F_CSKB1
        MOVX            A,@DPTR         ;czytaj klawiaturę
        CPL     A
        ANL     A,#00110000B            ;selekcjonuj tylko bity odpowiadające
                                        ;klawiszom strzałki w dół i strzałki w górę
        JNZ     JEST                    ;czy wciśnięto któryś z klawiszy strzałek?
                                        ;nie podano komendy,
                                        ;nie zmieniaj stanu F0 i PSW.1
        RET                             ;zakończ

JEST:                                   ;przyciśnięto klawisz strzałki
        CJNE    A,#20H,ZAPALAC          ;czy strzałka w dół (tj. "ZGAŚ")?
GASIC:
        JNB     F0,WY_1A                ;zmiana stanu
        JB      PSW.1,WY_1B             ;zmiana z "ZAPALANIE" na "GASZENIE"
        RET                             ;powtórne wciśnięcie klawisza, koniec proc.
WY_1A:
        JNB     PSW.1,POWROT            ;żądanie gaszenia, podczas gdy "ZGASZONE"

WY_1B:
        SETB    F0                      ;tak, zaznacz podając F0=1 i PSW.1=0 (=ZGAŚ)
        CLR     PSW.1
POWROT:
        RET                             ;zakończ

ZAPALAC:
        JNB     F0,WY_2A                ;zmiana stanu
        JNB     PSW.1,WY_2B             ;zmiana z "GAŚ" na "ZAPAL"
        RET                             ;powtórne wciśnięcie klawisza, koniec proc.
WY_2A:
        JB      PSW.1,POWROT            ;żądanie zapalenia, podczas gdy "ZAPALONE"

WY_2B:
        SETB    F0                      ;strzałka w górę, zaznacz podając F0=1
        SETB    PSW.1                   ;i PSW.1=1 (="ZAPAL")
```

```
        RET                         ;koniec


SKRACAJ:                            ;zmienia stan pocz. timera przech. w R7 i R6
                                    ;tak, aby opóźnienie wysterowania skracało się
        ..........
        ??????????
        ..........
        J??     DALEJ_0             ;czy skrócono maksymalnie? Jeżeli nie to skocz.
        CLR     F0                  ;koniec skracania opóźnienia, zaznacz podając
        SETB    PSW.1               ;F0=0 i PSW.1=1 ("ZAPALONE")
DALEJ_0:
        MOV     R6,B                ;zapamiętaj stan pocz. w R7 R6
        MOV     R7,A
        RET                         ;koniec procedury

        WYDLUZAJ:                   ;zmienia stan pocz. timera przech. w R7 i R6
                                    ;tak, aby opóźnienie wysterowania wydłużało się
        ..............
        ??????????????
        ..............

        ???     ???,???,DALEJ_1     ;czy zmniejszono wart. początkową
ZA_MALO:                            ;TH0 do minimalnej wartości? Skok jeżeli nie.
        MOV     A,#MIN_TH           ;tak, wpisz do AB min. wartość
        MOV     B,#MIN_TL

        CLR     F0                  ;koniec wydłużania opóźnienia, zaznacz podając
        CLR     PSW.1               ;F0=0 i PSW.1=0 ("ZGASZONE")
        CLR     C
DALEJ_1:
        JC      ZA_MALO             ;zmniejszono poniżej wart. minimalnej
        MOV     R6,B                ;zapamiętaj stan pocz. w R7 R6
        MOV     R7,A
        RET                         ;koniec procedury
```

ĆWICZENIE 6

Transmisja równoległa. Model: Skrzyżowanie.

W systemach mikroprocesorowych często zachodzi konieczność dobudowania kilku portów wejść/wyjść. Zamiast stosowania pojedynczych układów i mozolnego rozprowadzania szyny danych po płytce drukowanej oraz budowania dekodera adresów, można podłączyć układ 8255. Zawiera on w sobie 3 porty, które mogą być indywidualnie konfigurowane. Oprócz pełnienia typowych funkcji portu wejściowego bądź wyjściowego, mogą one również spełniać funkcję układu realizującego transmisję równoległą. W tym trybie następuje automatyczne potwierdzanie przesłanych bajtów i zgłaszanie, w odpowiednich momentach, przerwań do mikroprocesora.
W systemie DSM-51 został umieszczony układ 8255, choć nie jest on wykorzystywany do sterowania wewnętrznych elementów systemu. Wszystkie jego trzy porty zostały podłączone do złącza wejść/wyjść cyfrowych. Jest on więc przeznaczony do sterowania różnorodnych układów podłączanych do DSM-51.
W strukturze DSM-51 układ 8255 podłączony jest do szyny systemowej mikrokontrolera. Oprócz podłączenia szyny danych, sygnałów RD, WR, sygnału wyboru z dekodera adresów – CS55, są do niego dołączone dwie linie adresowe: A0 i A1. Układ 8255 zajmuje w przestrzeni adresowej mikrokontrolera cztery kolejne adresy. W systemie DSM-51 jego rejestry występują pod nazwami: CS55A, CS55B, CS55C, CS55D. Pierwsze trzy z tych rejestrów to w rzeczywistości trzy porty nazywane tutaj A, B i C. Czwarty rejestr jest rejestrem sterującym. Wpisanie odpowiedniego bajtu do tego rejestru ustawia tryby pracy portów. Tryby te przedstawione są w niniejszej lekcji.
Ponieważ porty układu 8255 podłączone są do złącza wejść/wyjść cyfrowych, przy omawianiu tego układu posłużono się zewnętrznymi urządzeniami sterowanymi przez system DSM-51. W tym celu wykorzystano opracowane w firmie MicroMade modele.
Najprostszym modelem podłączonym do złącza wejść/wyjść cyfrowych, a więc do układu 8255, jest model M-01 przedstawiający skrzyżowanie dróg. Poniżej przedstawiony jest schemat blokowy tego modelu.
W modelu M-01 wykorzystano jedynie częściowo port A i port B. Oba te porty powinny pracować jako wyjściowe, gdyż ich zadaniem jest zapalenie odpowiednich diod świecących symbolizujących światła na skrzyżowaniu ulic. Jak widać, do portu A przyporządkowano światła dla pieszych, natomiast do portu B światła dla samochodów. Faktycznie linia w porcie wysterowuje kilka diod na skrzyżowaniu, które pełnią tę samą funkcję. Na przykład w omawianym modelu są aż cztery czerwone diody dla przejść pionowych, sterowane przez linię PA0. Wszystkie one zapalane są i gaszone jednocześnie sygnałem z tej linii. Stan 0 na odpowiedniej linii zaświeca przyporządkowane do niej diody, natomiast stan 1 je gasi.



Prawidłowy program obsługi tego modelu polega na cyklicznym zapalaniu odpowiednich świateł na skrzyżowaniu, zgodnie z ogólnie znaną logiką. Poniżej przedstawiony jest przykład ilustrujący prawidłową obsługę układu 8255.

; **********Ustawienie 8255**********
;PORT A - > Światła dla pieszych
; PA0 - > przejście pionowo czerwone

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

```
; PA1 - >                    zielone
; PA2 - > przejście poziomo czerwone
; PA3 - >                    zielone

; PORT B - > Światła dla samochodów
; PB3 - > przejazd pionowo czerwone
; PB4 - >                    żółte
; PB5 - >                    zielone
; PB0 - > przejazd poziomo czerwone
; PB1 - >                    żółte
; PB2 - >                    zielone

PA_M EQU   0                          ; TRYB       0..2
PA_D  EQU  0                          ; OUT->      0, IN->1
PCA_D      EQU   0                    ; OUT->      0, IN->1
PB_M EQU   0                          ; TRYB 0..1
PCB_D      EQU   0                    ; OUT->      0, IN->1

PA     EQU   PA_M*4+PA_D*2+PCA_D
PB     EQU   PB_M*4+PB_D*2+PCB_D
SET_8255     EQU   80H+PA*8+PB

;*************************************
       LJMP  START

;*************************************
       ORG   100H
START:
       MOV   R0, #CS55D               ; inicjalizacja 8255
       MOV   A, #SET_8255
       MOVX        @R0, A

       MOV   R0, #CS55A               ; zgaszenie świateł
       MOV   A, #OFFH                 ; -wpisanie jedynek
       MOVX        @R0,A              ; na port A i B
       INC   R0
       MOVX        @R0,A

       MOV   R6,A                     ; stan LEDów
LOOP:
       MOV   R7,#6                    ; licznik – liczba diod
LOOP_ON:                              ; pętla włączania diod
       MOV   A,#5
       LCALL       DELAY_100MS
       MOV   A,R6
       CLR   C                        ; zapalenie kolejnej diody
       RLC   A
       MOV   R6,A
       MOVX        @R0,A
```

```
        DJNZ  R7,LOOP_ON            ; powtórz zgodnie z R7
        MOV   R7,#6
LOOP_OFF:                           ; pętla wyłączania diod
        MOV   A,#5
        LCALL       DELAY_100MS
        MOV   A,R6
        SETB  C                     ; zgaszenie kolejnej diody
        RLC   A
        MOV   R6,A
        MOVX        @R0,A
        DJNZ  R7,LOOP_OFF           ; powtórz zgodnie z R7
        SJMP  LOOP
```

Przed przystąpieniem do wystawiania odpowiednich danych na porty, należy odpowiednio ustawić rejestr sterujący. W tym celu należy wpisać pod adres CS55D bajt, w którym poszczególne bajty odpowiednie znaczenie.
Oto bajt sterujący pracą układu 8255
Bity:

7 – musi być =1
6,5 – ustawiają tryb portu A (00 – Tryb 0, 01 – Tryb 1, 1x – Tryb 2)
4 – ustawia port A jako wejście lub wyjście (0 – wyjście, 1 – wejście)
3 – ustawia starsze 4 bity portu C jako wejście/wyjście (0 – wyjście, 1 – wejście)
2 – ustawia tryb portu B
1 – ustawia port B jako wejście lub wyjście (0 – wyjście, 1 – wejście)
0 – ustawia młodsze 4 bity portu C jako wejście/wyjście (0 – wyjście, 1 – wejście)

We współpracy z modelem M-01 oba używane porty (A i B) muszą pracować w trybie 0 jako porty wyjściowe. W trybie tym porty pracują tak, jak zwykłe porty wyjściowe podłączone do szyny mikrokontrolera, tj. tak jak omawiane w lekcji 7 porty sterujące pracą wyświetlacza 7-segmentowego. Na liniach wyjściowych pojawiają się stany zgodne z wartością poszczególnych bitów bajtu wpisanego do portu. Stan ten jest utrzymywany aż do następnego wpisu. Zgodnie z tymi wymaganiami, w powyższym przykładzie do rejestru sterującego został wpisany bajt 10000000B. W rzeczywistości bity 0 i 3, sterujące portem C, mogą być ustawione dowolnie, gdyż port ten nie jest wykorzystywany w tym modelu.
Po zapisie bajtu do rejestru sterującego można przystąpić do wpisywania danych do portów. Co 0,5s jest wpisywany nowy bajt do portu B. W ten sposób co 0,5s następuje zmiana świecenia się diod sterujących przejazdem samochodów. Kolejno zapalane są diody, począwszy od podłączonych do linii PB0 a kończąc w momencie, gdy świecą się wszystkie diody podłączone do portu B. Potem następuje kolejne gaszenie diod.
Prawidłowe sterowanie tego modelu opiera się na prostym wpisywaniu kolejnych danych do portów A i B, zgodnie z założonym algorytmem sterowania światłami.

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

Wykonanie ćwiczenia

ZADANIE 1

W celu zapoznania się z trybami pracy układu 8255 oraz ze strukturą modelu M-01 należy prześledzić pracę programu z przykładu L16_1.
W trybie pracy krokowej odczytywać zawartość rejestru R7, ACC, R6 i obserwować zapalane i gaszone diody.

Lp.	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	R6	R7

```
;*********************************************
;LEKCJA  16  - UKŁAD TRANSMISJI RÓWNOLEGłEJ 8255
;PRZYŁAD 1 - TRYB 0 - PORT A i PORT B
;*********************************************

;****************** Ustawienie 8255 ***************
;PORT A              - światła dla pieszych
;PA0                 - przejście pionowo czerwone
;PA1                 -                  zielone
;PA2                 - przejście poziomo czerwone
;PA3                 -                   zielone

;PORT B              - światła dla samochodów
;PB3                 - przejazd pionowo czerwone
;PB4                 -                   żółte
;PB5                 -                   zielone
;PB0                 - przejazd poziomo czerwone
;PB1                 -                  żółte
;PB2                 -                  zielone

PA_M        EQU  0              ;TRYB 0..2
PA_D        EQU  0              ;OUT->0, IN->1
PCA_D       EQU  0              ;OUT->0, IN->1
PB_M        EQU  0              ;TRYB 0..1
PB_D        EQU  0              ;OUT->0, IN->1
PCB_D       EQU  0              ;OUT->0, IN->1

PA          EQU  PA_M*4+PA_D*2+PCA_D
PB          EQU  PB_M*4+PB_D*2+PCB_D
SET_8255    EQU  80H+PA*8+PB

;*************************************
        LJMP  START
```

```
;*********************************
        ORG   100H
START:
        MOV   R0,#CS55D                 ;inicjalizacja 8255
        MOV   A,#SET_8255
        MOVX        @R0,A

        MOV   R0,#CS55B                 ;zgaszenie świateł
        MOV   A,#0FFH                   ;-wpisanie jedynek
        MOVX        @R0,A               ;na port A i B
        DEC   R0
        MOVX        @R0,A


        MOV   R6,A                      ;stan LEDów
        INC   R0
LOOP:

        MOV   R7,#6                     ;licznik - liczba diod
LOOP_ON:                                ;pętla włączania diod
        MOV   A,#5
        LCALL       DELAY_100MS

        MOV   A,R6
        CLR   C                         ;zapalenie kolejnej diody
        RLC   A
        MOV   R6,A
        MOVX        @R0,A
        DJNZ  R7,LOOP_ON                ;powtórz zgodnie z R7

        MOV   R7,#6
LOOP_OFF:                               ;pętla wyłączania diod
        MOV   A,#5
        LCALL       DELAY_100MS

        MOV   A,R6
        SETB  C                         ;zgaszenie kolejnej diody
        RLC   A
        MOV   R6,A
        MOVX        @R0,A
        DJNZ  R7,LOOP_OFF               ;powtórz zgodnie z R7

        SJMP  LOOP
```

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ZADANIE 2

Zmodyfikować przykład z zadania 1 tak, by dla świateł dla pieszych wykonywał analogiczny test jak w zadaniu 1.
W trybie pracy krokowej odczytywać zawartość rejestru R7, ACC, R6 i w pętli włączania a następnie w pętli wyłączania obserwować zapalane i gaszone diody. Zwróć uwagę na wykorzystanie akumulatora!

Lp.	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	R6	R7

Zadanie 3.

Wykonać w trybie krokowym program skrzyż.hex. Analiza programu jak w zadaniu 2.

<u>UWAGA!</u>
Do procedur SET_AUTO, SET_PEOPLE, SET_BLINK, SET_TIME przekazywany jest poprzez akumulator parametr – numer stanu świateł (patrz opis). Na wyjściu z każdej z procedur w akumulatorze przekazywany jest kod binarny aktywacji poszczególnych LED-ów stosownie do stanu świateł skrzyżowania.

```
;TITLE          'DSM51-M01 SKRZYŻOWANIE'
;*************************************
;Program ustawia 8 kolejnych stanów świateł na skrzyżowaniu
;1 - przejazd samochodów poziomo (i przejście poziomo pieszych)
;2 - zmiana świateł - żółte dla samochodów jadących poziomo
;                   - zielone mrugające dla pieszych
;3 - zmiana świateł - czerwone dla samochodów jadących poziomo
;                   - czerwone dla pieszych
;4 - zmiana świateł - czerwone z żółtym dla samochodów w pionie
;5 - przejazd samochodów pionowo (i przejście pionowo pieszych)
;6 - zmiana świateł - żółte dla samochodów jadących pionowo
;                   - zielone mrugające dla pieszych
;7 - zmiana świateł - czerwone dla samochodów jadących pionowo
;                    - czerwone dla pieszych
;8 - zmiana świateł - czerwone z żółtym dla samochodów w poziomie

;*************************************
;Ustawienie układu 8255
;PORT A - WYJŚCIE MOD 0 - świata dla pieszych
;przejście pionowo
;A0 -> czerwone
;A1 -> zielone
;przejście poziomo
;A2 -> czerwone
;A3 -> zielone
```

```
;PORT B - WYJŚCIE MOD 0 - światła dla samochodów
;przejazd pionowo
;B3     - czerwone
;B4     - żółte
;B5     - zielone
;przejazd poziomo
;B0     - czerwone
;B1     - żółte
;B2     - zielone

;PORT C - WYJŚCIE - nieużywane

SET_8255      EQU   10000000B

;*************************************
        LJMP  START

;*************************************
        ORG   100H
START:
        MOV         R0,#CS55D               ;inicjalizacja 8255
        MOV         A,#SET_8255
        MOVX        @R0,A

        MOV         R0,#CS55A               ;port A - światła dla pieszych
        MOV         R1,#CS55B               ;port B - światła dla samochodów

;*************************************
;kolejne powtórzenie wszystkich stanów świateł na skrzyżowaniu
LOOP:
        MOV         R7,#8                   ;8 stanów świateł na skrzyżowaniu
        MOV         R2,#1                   ;stan pierwszy

;*************************************
;ustawienie kolejnego stanu świateł
STAN:
        MOV         A,R2                    ;światła dla samochodów
        ACALL       SET_AUTO                ;dla stanu numer (R2)
        MOVX        @R1,A

        MOV         A,R2                    ; światła dla pieszych
        ACALL       SET_PEOPLE              ;dla stanu numer (R2)
        MOVX        @R0,A
        MOV         R3,A                    ;zapamiętaj status pieszych

        MOV         A,R2                    ;mruganie światła dla pieszych
        ACALL       SET_BLINK               ;dla stanu numer (R2)
        MOV         R4,A                    ;zapamiętaj status mrugania
```

```
        MOV     A,R2                    ;czas świateł w stanie numer (R2)
        ACALL   SET_TIME                ;N * 0.5 sek
        MOV     R5,A                    ;zapamiętaj czas

        MOV     A,R2                    ;text na wyświetlacz LCD
        ACALL   SET_TEXT                ;dla stanu numer (R2)
        LCALL   LCD_CLR
        LCALL   WRITE_TEXT

;*************************************
;odliczanie czasu jednego stanu z dokładnością 0.5 sek
;mruganie światła dla pieszych jeżeli to konieczne
;z częstotliwością 1Hz
BLINK:
        MOV     A,#5
        LCALL   DELAY_100MS

        MOV     A,R3                    ;mruganie świateł dla pieszych
        XRL     A,R4                    ;-zmiana stanu na przeciwne
        MOV     R3,A                    ;dla wybranych świateł
        MOVX    @R0,A

        DJNZ    R5,BLINK                ;czas = R5 * 0.5 sek

        INC     R2                      ;kolejny stan
        DJNZ    R7,STAN

        SJMP    LOOP                    ;rozpocznij od pierwszego stanu

;*************************************
;dane do zapalenia świateł dla samochodów w 8 kolejnych stanach
SET_AUTO:
        MOVC    A,@A+PC
        RET
        DB      11110011B,11110101B,11110110B,11100110B
        DB      11011110B,11101110B,11110110B,11110100B

;*************************************
;dane do zapalenia świateł dla pieszych w 8 kolejnych stanach
SET_PEOPLE:
        MOVC    A,@A+PC
        RET
        DB      11110110B,11110110B,11111010B,11111010B
        DB      11111001B,11111001B,11111010B,11111010B

;*************************************
;dane do mrugania świateł dla pieszych w 8 kolejnych stanach
;1-mruganie odpowiedniego światła
```

```
SET_BLINK:
        MOVC        A,@A+PC
        RET
        DB      00000000B,00001000B,00000000B,00000000B
        DB      00000000B,00000010B,00000000B,00000000B

;*************************************
;czas kolejnych stanów wyrażony w 0.5 sek
SET_TIME:
        MOVC        A,@A+PC
        RET
        DB      15,8,4,4,15,8,4,4

;*************************************
;pobranie adresu tekstu dla kolejnego stanu
SET_TEXT:
        RL          A
        PUSH        ACC
        ACALL       SET_TXT
        MOV         DPL,A

        POP         ACC
        DEC         A
        ACALL       SET_TXT
        MOV         DPH,A
        RET

;*************************************
SET_TXT:
        MOVC        A,@A+PC
        RET

        DW      'TEXT1,TEXT2,TEXT3,TEXT4'
        DW      'TEXT5,TEXT2,TEXT3,TEXT4'

;*************************************
;teksty opisujące stan na skrzyżowaniu w kolejnych stanach
TEXT1:
        DB      'PRZEJAZD POZIOMO',0

TEXT2:
        DB      'ZMIANA SWIATEL  '
        DB      'ZOLTE ',0

TEXT3:
        DB      'ZMIANA SWIATEL  '
        DB      'CZERWONE',0

TEXT4:
```

```
        DB      'ZMIANA SWIATEL  '
        DB      'CZERWONE ZOLTE',0

TEXT5:
        DB      'PRZEJAZD PIONOWO',0
;*************************************
;END
```

W przykładowej tabeli należy rozpisać i zdekodować tabelę stanów świateł na skrzyżowaniu zgodnie z procedurą SET_AUTO dla samochodów oraz SET_PEOPLE dla pieszych.

STAN ŚWIATEŁ	PORT A	STAN	ZNACZENIE	WŁĄCZONE/ WYŁĄCZONE
	B7			
	B6			
	B5			
1	B4			
	B3			
	B2			
	B1			
	B0			
	B7			
	B6			
	B5			
2	B4			
	B3			
	B2			
	B1			
	B0			

…

	B7			
	B6			
	B5			
8	B4			
	B3			
	B2			
	B1			
	B0			

Zadanie 4.

Zmodyfikuj program z zadania 3 tak, by czas pomiędzy kolejnymi zmianami świateł odmierzany był za pomocą TIMERA. Należy przyjąć:
- korzystamy z TIMERA 0, koniec odcinka czasu rozpoznajemy ustawieniem flagi TF0,
- czas przejazdu w kierunku I jest wyróżniony – trwa dwukrotnie dłużej niż w kierunku II

Uwaga! – modyfikacji podlegają procedury BLINK, SET_TIME.
Prześledzić program w trybie pracy krokowej.

ĆWICZENIE 7.

SYSTEM PRZERWAŃ MIKROKONTROLERA 8051. UKŁADY CZASU RZECZYWISTEGO.

Wykonaj następujące zadania:

ZADANIE 1

Posługując się edytorem tekstowym uzupełnij zapisane w pliku **przer01.asm** program główny oraz procedury obsługi przerwań w taki sposób, aby program przyjmował i obsługiwał następujące przerwania:

a) Timera 0
b) Timera 1
c) Timera 0 i INT 1 (przetwornik A/C)

Każdorazowo wykonaj asemblację programu uzupełnionego zgodnie z wymaganiami punktów a), b), c) i uruchom program na DSM-51.

Po uzyskaniu poprawnej pracy programów zanotuj odpowiednie programy źródłowe i zamieść je w sprawozdaniu. Należy pominąć treść procedur INICJALIZACJA, JEST_T0, JEST_T1, JEST_I1.

Występująca w programie głównym procedura INICJALIZACJA czyści wyświetlacz LCD, programuje timery 0 i 1 w trybie 1 (timery 16 bitowe) i ustala stany początkowe timerów na 0000 i uruchamia odliczanie czasu, ponadto uruchamia przetwornik A/C. W wyniku wyżej opisanej inicjalizacji po ok. 150 μs przetwornik A/C zgłasza przerwanie za pomocą linii INT1, natomiast obydwa timery zgłaszają przerwania wewnętrzne po upływie ok. 71 ms.

Procedura JEST_T0 wysyła na wyświetlacz LCD znaki „T0” oraz zatrzymuje Timer 0 (bit TR0).
Procedura JEST_T1 wysyła na wyświetlacz LCD znaki „T1” oraz zatrzymuje Timer 1 (bit TR1).
Procedura JEST_I1 wysyła na wyświetlacz LCD znaki „I1”.

ZADANIE 2

Program **przer02.hex** działa analogicznie jak program w zadaniu 1 z tym, że reaguje wyłącznie na przerwania od timerów. Timery uruchamiane są razem i od ustawionych stanów początkowych zliczają impulsy zegara systemowego. W momencie przepełnienia się rejestrów timera (stan FFFF) następuje wysłanie przerwania. Czas upływający od momentu uruchomienia programu do momentu wysłania przerwania przez dany timer jest tym krótszy, im większą wartość początkową ustawi się w timerze.

Przeanalizuj program główny z pliku **przer02.asm**. Posługując się edytorem tekstowym wstawiaj w pliku **przer02.asm** wartości początkowe dla timerów 0 i 1 zgodnie z poniższą tabelą. Uzyskany program poddawaj asemblacji i uruchamiaj. Każdorazowo

zanotuj kolejność pojawiania się komunikatów o obsłudze danego przerwania. Obsługa przerwania od Timera 0 sygnalizowana jest znakami „T0”, Timera 1 znakami „T1”.

Lp.	Stan pocz. T0	Stan pocz. T1	Kolejność
1	0000	0000	
2	0000	0001	
3	0001	0000	

Wyjaśnij przyczyny zanotowanej kolejności obsługiwania przerwań.

ZADANIE 3

Działanie programu **przer03.hex** jest analogiczne do działania programu **przer01**, reaguje na przerwania z timerów 0 i 1 oraz linii INT1 (przetwornik A/C). Czas przetwarzania przetwornika A/C wynosi ok. 150 μs; po takim czasie od uruchomienia programu **przer03.hex** przetwornik wysyła przerwanie na linię INT1. Dobierając odpowiednie wartości początkowe timerów 0 i 1 można powodować, że przerwanie każdego z timerów będzie poprzedzać bądź następować po przerwaniu przetwornika.

Posługując się edytorem tekstowym wstawiaj w pliku **przer03.asm** wartości początkowe dla timerów 0 i 1 zgodnie z poniższą tabelą. W ostatnim, 5 punkcie zablokuj przerwanie od Timera 0. Uzyskany program poddawaj asemblacji i uruchamiaj. Każdorazowo zanotuj kolejność pojawiania się komunikatów o obsłudze danego przerwania. Obsługa przerwania od Timera 0 sygnalizowana jest znakami „T0”, Timera 1 znakami „T1”, przetwornika A/C znakami „Y0”.

Lp.	Stan pocz. T0	Stan pocz. T1	Kolejność
1	FFFF	FFFF	
2	FF80	FF80	
3	FF40	FF40	
4	FF10	FF10	
5	zablokowany	FF40	

Wyjaśnij przyczyny zanotowanej kolejności obsługiwania przerwań.

ZADANIE 4

Program **przer04.hex** działa analogicznie jak program w zadaniu 2. Posługując się edytorem tekstowym wstawiaj w pliku **przer04.asm** wartości początkowe dla timerów 0 i 1 oraz bity priorytetów przerwań zgodnie z poniższą tabelą. Uzyskany program poddawaj asemblacji i uruchamiaj. Każdorazowo zanotuj kolejność pojawiania się komunikatów o obsłudze danego przerwania. Obsługa przerwania od Timera 0 sygnalizowana jest znakami „T0”, Timera 1 znakami „T1”.

	Timer 0		Timer 1		
Lp	Stan pocz.	Pr.	Stan pocz.	Pr.	Kolejność
1	0000	0	0000	0	

2	0000	0	0000	1	
3	0000	0	0001	0	
4	0000	1	0001	0	
5	0000	1	0001	1	

Wyjaśnij przyczyny zanotowanej kolejności obsługiwania przerwań.

ZADANIE 5

Uruchom na DSM-51 program **przer05.hex.** Zadaniem programu jest włączanie i wyłączanie LED co okres odmierzany w timerze przy zastosowaniu przerwań.
Przeanalizuj, a następnie skomentuj listing programu.

ZADANIE 6

Uruchom na DSM-51 program **przer06.hex.** Zadaniem programu jest pobieranie znaku 0..9 z klawiatury i wysyłanie na wyświetlacz oraz jednoczesne (w tle) włączanie i wyłączanie LED co okres odmierzany w timerze.

Sprawdź, czy program prawidłowo wyprowadza znaki na wyświetlacz. W jaki sposób może tu dochodzić do wpływania na siebie programu i procedury obsługi programu. Przeanalizuj listing programu pod tym kątem.

ZADANIE 7
(dla zaawansowanych)

W programie **przer04.asm** odblokuj przerwanie z linii INT1 (przetwornik A/C), ustaw stany początkowe timerów 0 i 1 na FF40H (przy takiej wartości początkowej każdy z timerów wyśle przerwanie po upływie ok. 150 µs od uruchomienia programu). Zaproponuj ustawienie priorytetów aby przy jednoczesnym zgłoszeniu przerwań od Timera 0, Timera 1 i linii INT1 pierwsze zostało obsłużone przerwanie z Timera 1, następnie z Timera 0 i na końcu z INT1. W celu sprawdzenia propozycji ustaw priorytety w programie **przer04.asm**, zasembluj i uruchom program.

Czy w przypadku jednoczesnego zgłoszenia przerwań jak wyżej jest możliwa obsługa przerwań w następującej kolejności: Timer 1, INT1, Timer 0?

ZADANIE 8
(dla zaawansowanych)

Zaproponuj wykorzystanie stosu do usunięcia trudności występujących w programie **przer06.hex**. Zmodyfikuj, zasembluj uruchom program. Program źródłowy zawarto w pliku **przer06.asm**.

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

LISTINGI

```
;*********************************************
;ĆWICZENIE 7        - SYSTEM PRZERWAŃ 8051
;ZADANIE   1        - ŹRÓDŁA PRZERWAŃ W 8051
;*********************************************

;******** Ustawienie TIMERÓW ********
;TIMER 0
T0_G  EQU   0                       ;GATE
T0_C  EQU   0                       ;COUNTER/-TIMER
T0_M  EQU   1                       ;MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8      ;TIMER 1
T1_G  EQU   0                       ;GATE
T1_C  EQU   0                       ;COUNTER/-TIMER
T1_M  EQU   1                       ;MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET   EQU   TIM0+TIM1*16

TH0_SET            EQU   00H        ;stan początkowy Timera 0. Starszy bajt.
TL0_SET            EQU   00H        ;Młodszy bajt.
TH1_SET            EQU   00H        ;stan początkowy Timera 1. Starszy bajt.
TL1_SET            EQU   00H        ;Młodszy bajt.

;************************************
        LJMP  START
;******** Przerwanie Timer 0 ********
        ORG   ???
        LCALL       JEST_T0
        ???
;******** Przerwanie INT1 (AC) ********
        ORG   ???
        LCALL       JEST_I1
        ???
;******** Przerwanie Timer 1 ********
        ORG   ???
        LCALL       JEST_T1
        ???

;*****************************************************************
;
;       Program główny
;
;*****************************************************************
        ORG   100H
START:
        LCALL       INICJALIZACJA           ;inicjalizacja źródeł przerwań
        ???     ???                 ;???
```

```
        ...................................
        ???     ???                                     ;???
LOOP:
        SJMP  LOOP                                      ;koniec programu głównego

;*****************************************************
;
;       Podprogramy do użycia w procedurach obsługi przerwań
;       UWAGA! nie przepisywać do sprawozdania
;
;*************************************************************
```

***

```
;*********************************************
;ĆWICZENIE 7        - SYSTEM PRZERWAŃ 8051
;ZADANIE   2        -
;*********************************************

;********* Ustawienie TIMERÓW *********
;TIMER 0
T0_G  EQU   0                           ;GATE
T0_C  EQU   0                           ;COUNTER/-TIMER
T0_M  EQU   1                           ;MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8          ;TIMER 1
T1_G  EQU   0                           ;GATE
T1_C  EQU   0                           ;COUNTER/-TIMER
T1_M  EQU   1                           ;MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

TH0_SET         EQU   0FFH      ;stan początkowy Timera 0. Starszy bajt.
TL0_SET         EQU   028H      ;Młodszy bajt.
TH1_SET         EQU   0FFH      ;stan początkowy Timera 1. Starszy bajt.
TL1_SET         EQU   028H      ;Młodszy bajt.

REJ0        EQU   0
DPTH        EQU   083H
DPTL        EQU   082H
;*************************************
        LJMP  START
;********* Przerwanie INT0 *********
;********* Przerwanie Timer 0 *********
        ORG   0BH
        LJMP  TIMER0_INT
;********* Przerwanie INT1 (AC) *********
;********* Przerwanie Timer 1 *********
        ORG   1BH
        LJMP  TIMER1_INT
```

```
;********* Przerwanie Transm. szer. *********
;*****************************************************************
;
;       Robocza część programu głównego
;
;*****************************************************************
        ORG   100H
START:
        LCALL       INICJALIZACJA           ;inicjalizacja źródeł przerwań
        SETB  ET1                           ;zezwolenie na przerwanie z timera 1
        SETB  ET0                           ;zezwolenie na przerwanie z timera 0
        SETB  EA                            ;ogólne zezwolenie na przerwanie
LOOP:
        LCALL       LCD_CLR                 ;czyść wyświetlacz
        MOV   DPTR,#POLE                    ;wyświetl zawartość
        LCALL       WRITE_TEXT              ;pola tekstu
        CPL   P1.7                          ;"przełącz" LED
        MOV   A,#0AH                        ;opóźnienie
        LCALL       DELAY_100MS             ;1 sek.
        SJMP  LOOP                          ;koniec programu głównego

POLE:                                       ;pole tekstu wyświetlanego na LCD
        DB    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

;*****************************************************************
;
;       Procedura inicjalizacji źródeł przerwań
;
;*****************************************************************
;*****************************************************************
;
;       Procedury obsługi przerwań
;
;*****************************************************************
```

***

```
;*********************************************
;ĆWICZENIE 7        - SYSTEM PRZERWAŃ 8051
;ZADANIE   5        -
;*********************************************
LED   EQU   P1.7

;********* Ustawienie TIMERÓW *********
;TIMER 0
T0_G  EQU   0                       ;GATE
T0_C  EQU   0                       ;COUNTER/-TIMER
T0_M  EQU   1                       ;MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8      ;TIMER 1
```

```
T1_G  EQU   0                           ;GATE
T1_C  EQU   0                           ;COUNTER/-TIMER
T1_M  EQU   0                           ;MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

TH0_SET           EQU   256-180
TL0_SET           EQU   0
;*************************************
        LJMP  START
;******** Przerwanie Timer 0 ********
        ORG   0BH
        MOV   TH0,#TH0_SET              ;
        DJNZ  ACC,NO_1SEK               ;
        CPL   LED                       ;
        MOV   A,#20                     ;
NO_1SEK:                                ;
        RETI
;*************************************
        ORG   100H
START:
        MOV   TMOD,#TMOD_SET            ;
        MOV   TH0,#TH0_SET              ;
        MOV   TL0,#TL0_SET              ;
        SETB  TR0                       ;
        MOV   A,#20                     ;
                                        ;
        SETB  EA                        ;
        SETB  ET0                       ;
        SJMP  $                         ;
```

***

```
;**********************************************
;ĆWICZENIE 7        - SYSTEM PRZERWAŃ 8051
;ZADANIE 6  -
;**********************************************
LED   EQU   P1.7
;******** Ustawienie TIMERÓW ********
;TIMER 0
T0_G  EQU   0                           ;GATE
T0_C  EQU   0                           ;COUNTER/-TIMER
T0_M  EQU   1                           ;MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8          ;TIMER 1

T1_G  EQU   0                           ;GATE
T1_C  EQU   0                           ;COUNTER/-TIMER
T1_M  EQU   0                           ;MODE (0..3)
```

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

```
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

TH0_SET         EQU   0EAH        ;stan początkowy Timera 0
TL0_SET         EQU   066H        ;opóźnienia 6ms
KROTNOSC  EQU   0FFH              ;krotność powtarzania opóźnień 6ms
;*************************************
        LJMP  START
;******** Przerwanie Timer 0 ********
        ORG   0BH
        MOV   TH0,#TH0_SET        ;stan pocz¹tkowy do TH0
        MOV   TL0,#TL0_SET        ;i do TL0 (6ms)
        DJNZ  B,JESZCZE_NIE       ;czy wykonano KROTNOSC opóźnień,
                                  ;tj. KROTNOSC*6ms?
        CPL   LED                 ;mruganie diody TEST
        MOV   B,#KROTNOSC         ;odczekaj kolejną KROTNOSC opóźnień
                                  ;KROTNOSC*6ms
JESZCZE_NIE:
        RETI
;*************************************
        ORG   100H
START:
        MOV   TMOD,#TMOD_SET      ;Timer 0 liczy czas
        MOV   TH0,#TH0_SET        ;stan początkowy do TH0
        MOV   TL0,#TL0_SET        ;i do TL0
        SETB  TR0                 ;start Timera 0
        SETB  EA                  ;włącz zezwolenie ogólne na przerwania
        SETB  ET0                 ;włącz zezwolenie na przerwanie od Timera 0
;*****************************************************************
;
;       Robocza część programu głównego
;
;*****************************************************************
LOOP_0:
        LCALL     LCD_CLR         ;czyść wyświetlacz
        MOV       R1,#16          ;do odlicz. max. 16 znaków na wyświetl.
LOOP:                             ;w tej pętli czyta/pisze znak
        LCALL     WAIT_KEY        ;czyta klawisz
        MOV       B,A             ;przechowuje znak
        MOV       A,#2            ;niewielkie
        LCALL     DELAY_MS        ;opóźnienie
        MOV   A,B                 ;pobiera znak z "przechowalni"
        ADD   A,#30H              ;zamienia binarny kod cyfry na kod w ASCI
        MOV   R0,#LCDWD           ;wysyła
        MOVX      @R0,A           ;znak na wyświetlacz

        DJNZ  R1,LOOP             ;wyczyść wyświetlacz, jeżeli było już 16 znaków
        SJMP  LOOP_0              ;zamknij pętlę
```

ĆWICZENIE 8

Układy przetworników A/C i C/A. Model: tester tranzystorów.

W wielu praktycznych zastosowaniach okazuje się, że mikrokontroler musi mieć możliwość pomiaru wielkości analogowych oraz sterowania takimi wielkościami. Dotyczy to w szczególności prawie każdego mikrokontrolera sterującego przebiegiem dowolnego procesu technologicznego. Parametrami procesów technologicznych są różnego rodzaju wielkości nieelektryczne. Wielkości takie, jak temperatura czy ciśnienie, są zamienione na sygnały elektryczne, a następnie przetwarzane na wartości liczbowe, ca pozwala na uzależnienie od nich procesu sterowania.
Do zamiany wielkości nieelektrycznych na elektryczne służą różnego rodzaju czujniki. Sygnał z czujnika jest najczęściej wzmacniany i zamieniany na napięcie przyjmujące wartości z pewnego określonego przedziału. Napięcie to jest z kolei zamieniane na liczbę określającą jego wartość. Zamiana ta nosi nazwę przetwarzania analogowo/cyfrowego, a elementy, które je wykonują to przetworniki analogowo/cyfrowe.
Na rynku dostępna jest szeroka gama przetworników. Podstawowymi parametrami charakteryzującymi przetworniki analogowo/cyfrowe są:

długość słowa: liczba bitów, na których podawany jest wynik (typowo 8, 10 lub 12)
czas przetwarzania: czas od rozpoczęcia przetwarzania do momentu, gdy wynik może być odczytany (od ułamków μs do setek ms)
zakres napięcia wejściowego: zakres napięcia podanego na wejście przetwornika, które zostanie prawidłowo przetworzone na odpowiadającą mu liczbę.

Do każdego konkretnego zadania dobiera się odpowiedni przetwornik w zależności od wymaganej dokładności przetwarzania i częstotliwości, z jaką należy daną wielkość kontrolować.
Przetworniki analogowo/cyfrowe wymagają zazwyczaj, aby mikrokontroler zainicjował proces przetwarzania i sygnalizują jego zakończenie. Mikrokontroler po odebraniu tego sygnału może odczytać z przetwornika wynik przetwarzania.
Gdy mikrokontroler ma sterować wielkościami analogowymi, zachodzi potrzeba przetwarzania w odwrotną stronę. Służą do tego przetworniki cyfrowo/analogowe. Sygnałem wyjściowym tych przetworników mogą być różne wielkości analogowe, takie jak napięcie, prąd czy wzmocnienie. Prostowniki cyfrowo/analogowe są prostsze w obsłudze od przetworników analogowo/cyfrowych gdyż wymagają jedynie wpisania wartości liczbowej, która ma być zamieniona na odpowiadającą jej wartość analogową.
Dydaktyczny System Mikroprocesorowy DSM-51 jest wyposażony w dwa popularne przetworniki 8-bitowe:
ADC0804 -przetwornik analogowo/cyfrowy,

DAC08 -przetwornik cyfrowo/analogowy.

System DSM-51 ma 8 wejść analogowych. Sygnały z tych wejść są podawane na multiplekser analogowy, który umożliwia podłączenie jednego z nich do wejścia przetwornika ADC0804.
Multiplekser jest sterowany buforem umieszczonym pod adresem 18H (CSMX) w przestrzeni adresowej urządzeń wejść/wyjść. Do bufora należy wpisać numer wejścia (0...7), które ma być podłączone do przetwornika. W danym momencie może być przetwarzana wartość

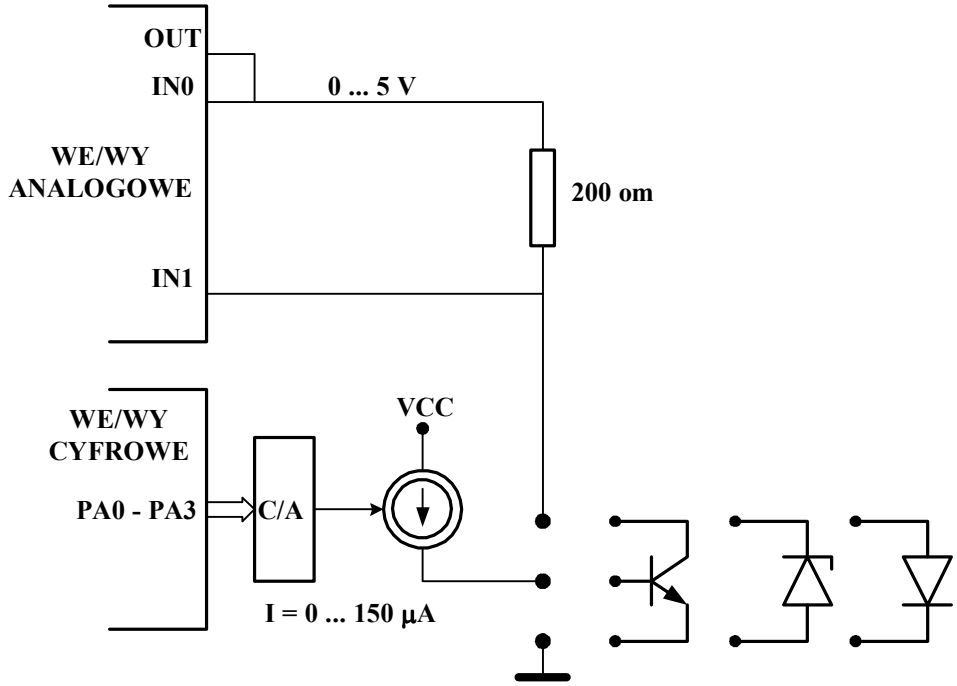
napięcia tylko z jednego wejścia. Wartościom napięcia z zakresu 0...5V odpowiadają wyniki przetwarzania z zakresu 00H...FFH (1 bajt). Przetwarzanie trwa około 150 μs.
Przetwornik jest umieszczony pod adresem 10H (CSAD) w przestrzeni adresowej urządzeń wejść/wyjść. Zapis dowolnej wartości pod ten adres powoduje start przetwarzania. W momencie zakończenia przetwarzania przetwornik generuje przerwanie. Wyjście przerwań przetwornika jest podłączone do systemu przerwań DSM-51 (sygnał IAD). Odczytanie wyniku przetwarzania (adres CSAD) można zrealizować po odczekaniu odpowiedniego czasu lub w wyniku reakcji na przerwanie.
System DSM-51 ma jedno wyjście analogowe. Bajt, który ma być przetwarzany na sygnał analogowy jest wpisywany do bufora, który następnie wysterowuje linie wejść cyfrowych przetwornika DAC08. Przetwornik steruje za pośrednictwem odpowiedniego wzmacniacza wyjściem analogowym systemu. Na wyjściu tym pojawia się napięcie z zakresu 0...5V odpowiadające wpisanej do bufora liczbie (00H...FFH). Bufor sterujący przetwornikiem ma adres 08H (CSDA).
Przykład 1 to prosty program obsługujący oba przetworniki systemu DSM-51.

```
        LJMP  START
        ORG   100H
START:

        MOV   R0,#CSMX          ;wybranie wejścia 0
        CLR   A
        MOVX        @R0,A
        MOV   R0,#CSDA          ;adres przetwornika C/A
        MOV   R1,#CSAD          ;adres przetwornika A/C

        LCALL       LCD_CLR

LOOP:
        LCALL       WAIT_KEY    ;wybrany klawisz (0..15)
        MOV   R2,A
        LCALL       LCD_CLR
        MOV   A,R2              ;powielenie numeru na
        SWAP A                  ;cały bajt
        ADD   A,R2
        MOVX        @R0,A       ;wpis do przetwornika C/A

        LCALL       WRITE_HEX   ;wpis na LCD wartości
        MOV   A,#'-'            ;przetwarzanej
        LCALL       WRITE_DATA
        MOV   A,#'>'
        LCALL       WRITE_DATA

        MOVX        @R1,A       ;inicjowanie pracy
        MOV   A,#1              ;przetwornika A/C
        LCALL       DELAY_MS
        MOVX        A,@R1       ;wynik przetwarzania A/C
        LCALL       WRITE_HEX   ;na wyświetlacz LCD
        SJMP  LOOP
```

Program ten po przełączeniu multipleksera na wejście analogowe numer 0 wykonuje w pętli następujące czynności

- czeka na naciśnięcie jednego z klawiszy klawiatury matrycowej,
- wpisuje do bufora sterującego przetwornikiem cyfrowo/analogowym bajt uzyskany przez wpisanie kodu klawisza do jego młodszej i starszej części,
- wykonuje pomiar przetwornikiem analogowo/cyfrowym,
- wypisuje na wyświetlaczu LCD obie wartości.

Aby sprawdzić działanie tego programu, należy wyjście analogowe połączyć z wejściem analogowym numer 0, czyli zacisk 4 (OUT) z zaciskiem 7 (IN0) złącza wejść/wyjść analogowych. Naciskając kolejne klawisze można zaobserwować, czy przetwarzanie jest prawidłowe. Przetworniki w systemie DSM-51 nie są kalibrowane, dlatego różnica między wartością wpisaną do przetwornika cyfrowo/analogowego a wartością odczytaną z przetwornika analogowo/cyfrowego może wynosić nawet kilka bitów.
Wśród przystawek do systemu DSM-51 znajdujących się w ofercie MicroMade jest tester diod i tranzystorów (model M-02).
Przystawka umożliwia pomiar charakterystyk prądowo-napięciowych diod półprzewodnikowych (również diod Zenera w kierunku przewodzenia i zaporowym) oraz rodziny charakterystyk wyjściowych tranzystorów n-p-n. Schemat blokowy przystawki przedstawiono na rysunku:



Model M-02 sterowany jest przez system DSM-51 za pośrednictwem dwu złącz: złącza wejść/wyjść cyfrowych oraz złącza wejść/wyjść analogowych.
Badany element zasilany jest z wyjścia przetwornika C/A systemu DSM-51 poprzez rezystor 200Ω. Napięcia z obu końców rezystora podane są do wejść analogowych (IN0, IN0) przetwornika A/C systemu. Pomiar tych dwu napięć pozwala ustalić zarówno napięcie panujące na badanym elemencie, jak i płynący przez ten element prąd.
Poszukiwana charakterystyka jest zależnością płynącego przez element prądu od panującego na nim napięcia.

Aby uzyskać zestaw punktów należących do tej charakterystyki, należy ustawić kolejne wartości napięcia na wyjściu przetwornika C/A i mierzyć napięcia panujące na obu końcach rezystora 200Ω.
Przykład 2 jest programem umożliwiającym pomiar takiej charakterystyki z wykorzystaniem modelu M-02.

```
        LJMP  START
        ORG   100H
START:

        MOV   R0,#CSDA              ;adres przetwornika C/A
        MOV   R1,#CSMX              ;adres multipleksera

        LCALL       LCD_CLR

LOOP:
        LCALL       WAIT_KEY        ;wybrany klawisz (0..15)
        MOV   R2,A
        LCALL       LCD_CLR
        MOV   A,R2                  ;powielenie numeru na
        SWAP A                      ;cały bajt
        ADD   A,R2
        MOVX        @R0,A           ;wpis do przetwornika C/A
        LCALL       WRITE_HEX       ;i wpis na LCD wartości
                                    ;podawanej na wyjście

        CLR   A                     ;podłączenie wejścia 0
        MOVX        @R1,A           ;do przetwornika A/C
        MOV   DPTR,#TEXT1
        LCALL       WRITE_TEXT
        DEC   R1                    ;inicjowanie pracy
        MOVX        @R1,A           ;przetwornika A/C
        MOV   A,#1
        LCALL       DELAY_MS
        MOVX        A,@R1           ;wynik pomiaru wejścia 0
        INC   R1
        LCALL       WRITE_HEX

        MOV   A,#1                  ;podłączenie wejścia 1
        MOVX        @R1,A           ;do przetwornika A/C
        MOV   DPTR,#TEXT2
        LCALL       WRITE_TEXT
        DEC   R1                    ;inicjowanie pracy
        MOVX        @R1,A           ;przetwornika A/C
        MOV   A,#1
        LCALL       DELAY_MS
        MOVX        A,@R1           ;wynik pomiaru wejścia 1
        INC   R1
        LCALL       WRITE_HEX
```

```
        SJMP  LOOP

TEXT1:
        DB    '-> N0=',0
TEXT2:
        DB    ' N1=',0
```

Wartość wpisywana do przetwornika C/A jest ustalona tak samo, jak w przykładzie 1. Po ustawieniu odpowiedniego napięcia na wyjściu analogowym następuje pomiar napięć na wejściach IN0 i IN1. Zmierzone wartości są wypisywane na wyświetlaczu LCD. Jeśli odczytane liczby wynoszą odpowiednio N0 i N1, to wartość napięcia panującego na badanym elemencie i płynącego prądu można uzyskać ze wzorów:

U = (N1 / 255) * 5V

I = {[(N0 – N1) / 255] * 5V}/200Ω

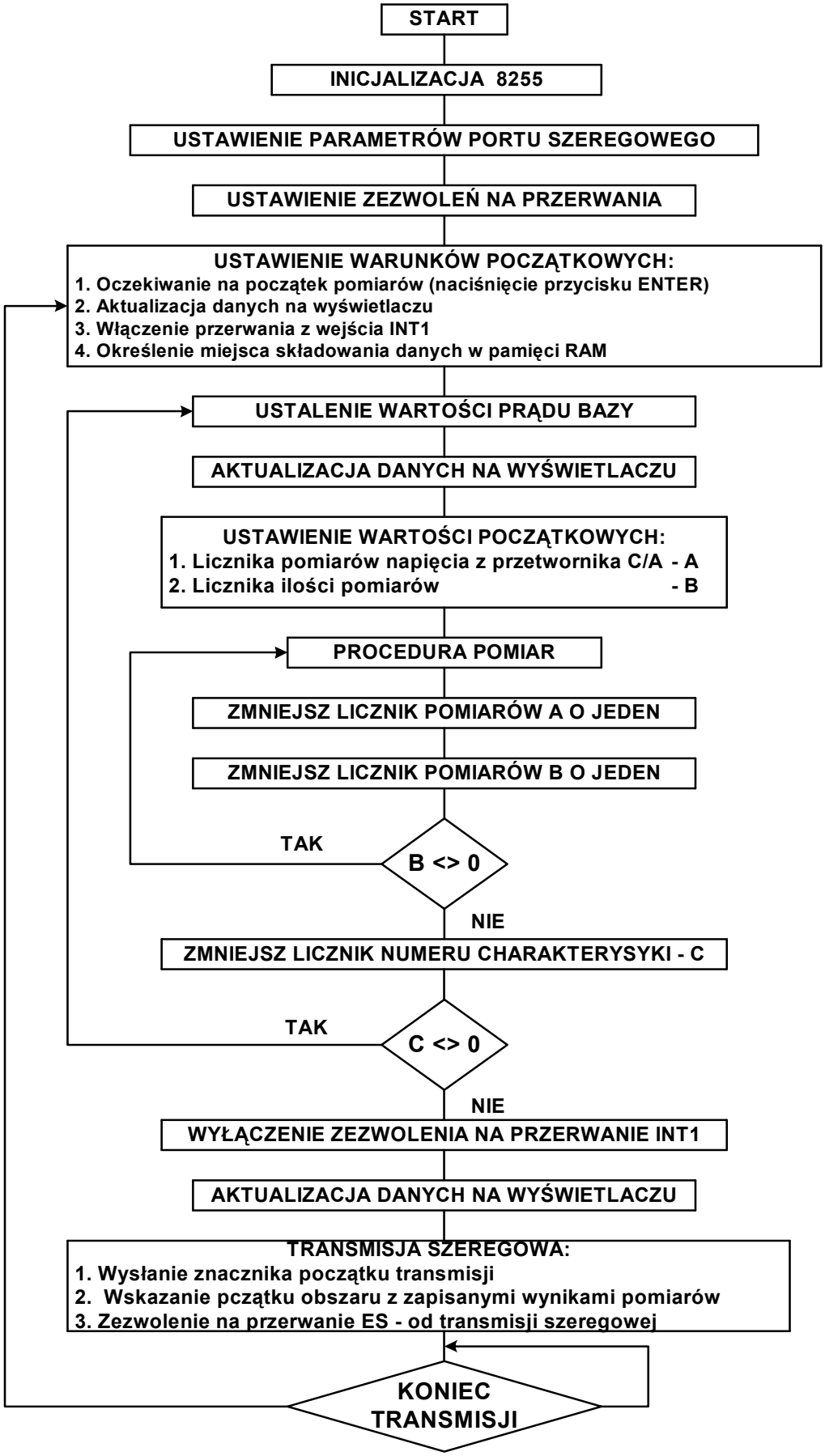
Rodzina charakterystyk wyjściowych tranzystora to zestaw charakterystyk prądowo-napięciowych złącza kolektor-emiter przy różnych wartościach prądu bazy. Pomiar tej rodziny charakterystyk jest możliwy dzięki umieszczeniu na przystawce sterowanego źródła prądowego zasilającego bazę badanego tranzystora

```
;********** Ustawienie 8255 ***********
;PORT A -> przetwornik C/A 4bit

PA_M EQU  0             ;TRYB 0..2
PA_D EQU  0             ;OUT->0, IN->1
PCA_D     EQU  0        ;OUT->0, IN->1
PB_M EQU  0             ;TRYB 0..1
PB_D EQU  0             ;OUT->0, IN->1
PCB_D     EQU  0        ;OUT->0, IN->1

PA    EQU  PA_M*4+PA_D*2+PCA_D
PB    EQU  PB_M*4+PB_D*2+PCB_D
SET_8255  EQU  80H+PA*8+PB

;************************************
        LJMP  START
        ORG   100H
START:

        MOV  R0,#CS55D              ;ustawienie układu 8255
        MOV  A,#SET_8255
        MOVX       @R0,A

        LCALL      LCD_CLR

        LCALL      WAIT_KEY         ;numer klawisza jako prąd
        MOV  R0,#CS55A              ;bazy tranzystora
        MOVX       @R0,A
```

```
        PUSH  ACC                       ;wpisz prąd bazy na LCD
        MOV   DPTR,#TEXT3
        LCALL         WRITE_TEXT
        POP   ACC
        LCALL         WRITE_HEX
        MOV   A,#8H+4H
        LCALL         WRITE_INSTR

        MOV   R0,#CSDA                  ;adres przetwornika C/A
        MOV   R1,#CSMX                  ;adres multipleksera
LOOP:
        LCALL         WAIT_KEY          ;wybrany klawisz (0..15)
        MOV   R2,A
        MOV   A,#80H+40H                ;LCD na początek drugiej
        LCALL         WRITE_INSTR       ;linii
        MOV   A,R2                      ;powielenie numeru na
        SWAP A                          ;cały bajt
        ADD   A,R2
        MOVX          @R0,A             ;wpis do przetwornika C/A
        LCALL         WRITE_HEX         ;i wpis na LCD wartości
                                        ;podawanej na wyjście

        CLR   A                         ;podłączenie wejścia 0
        MOVX          @R1,A             ;do przetwornika A/C
        MOV   DPTR,#TEXT1
        LCALL         WRITE_TEXT
        DEC   R1                        ;inicjowanie pracy
        MOVX          @R1,A             ;przetwornika A/C
        MOV   A,#1
        LCALL         DELAY_MS
        MOVX          A,@R1             ;wynik pomiaru wejścia 0
        INC   R1
        LCALL         WRITE_HEX

        MOV   A,#1                      ;podłączenie wejścia 1
        MOVX          @R1,A             ;do przetwornika A/C
        MOV   DPTR,#TEXT2
        LCALL         WRITE_TEXT
        DEC   R1                        ;inicjowanie pracy
        MOVX          @R1,A             ;przetwornika A/C
        MOV   A,#1
        LCALL         DELAY_MS
        MOVX          A,@R1             ;wynik pomiaru wejścia 1
        INC   R1
        LCALL         WRITE_HEX

        SJMP  LOOP
```

```
TEXT1:
        DB      '-> N0=',0
TEXT2:
        DB      ' N1=',0
TEXT3:
        DB      'Prąd Bazy = ',0
```

Po uruchomieniu programu pierwszy naciśnięty klawisz decyduje o wartości prądu bazy, przy którym są następnie wykonywane kolejne pomiary – jak w przykładzie 2. Aby pomierzyć charakterystykę tranzystora dla kolejnej wartości prądu bazy, należy wystartować program od początku ([RESET RAM]).
Przykładowe programy (DIODA.ASM i TRANZYST.ASM) demonstrujące sposób wykorzystania modelu M-02 znajdują się na dyskietce systemu DSM-51. Program DIODA>ASM pozwala na uzyskiwanie charakterystyk diod półprzewodnikowych (lub ich elementów dwukońcówkowych). Pomierzone dane są przesyłane przez łącze RS232 do pracującego na komputerze programu DIODA.EXE, który pokazuje na ekranie badaną charakterystykę. Podobnie para programów TRANZYST.ASM i TRANZYST.EXE pozwala na uzyskanie na ekranie komputera rodziny charakterystyk tranzystora n-p-n.


START
INICJALIZACJA 8255
USTAWIENIE PARAMETRÓW PORTU SZEREGOWEGO
USTAWIENIE ZEZWOLEŃ NA PRZERWANIA
USTAWIENIE WARUNKÓW POCZĄTKOWYCH:
1. Oczekiwanie na początek pomiarów (naciśnięcie przycisku ENTER)
2. Aktualizacja danych na wyświetlaczu
3. Włączenie przerwania z wejścia INT1
4. Określenie miejsca składowania danych w pamięci RAM
USTALENIE WARTOŚCI PRĄDU BAZY
AKTUALIZACJA DANYCH NA WYŚWIETLACZU
USTAWIENIE WARTOŚCI POCZĄTKOWYCH:
1. Licznika pomiarów napięcia z przetwornika C/A - A
2. Licznika ilości pomiarów - B
PROCEDURA POMIAR
ZMNIEJSZ LICZNIK POMIARÓW A O JEDEN
ZMNIEJSZ LICZNIK POMIARÓW B O JEDEN
TAK
B <> 0
NIE
ZMNIEJSZ LICZNIK NUMERU CHARAKTERYSYKI - C
TAK
C <> 0
NIE
WYŁĄCZENIE ZEZWOLENIA NA PRZERWANIE INT1
AKTUALIZACJA DANYCH NA WYŚWIETLACZU
TRANSMISJA SZEREGOWA:
1. Wysłanie znacznika początku transmisji
2. Wskazanie pczątku obszaru z zapisanymi wynikami pomiarów
3. Zezwolenie na przerwanie ES - od transmisji szeregowej
KONIEC TRANSMISJI

W niektórych przypadkach zastosowanie scalonego przetwornika cyfrowo/analogowego nie jest niezbędne. Inną metodą sterowania przez mikrokontroler wartością napięcia stałego jest wytwarzanie na jednym z wyjść przebiegu prostokątnego o określonym wypełnieniu. Po Podaniu takiego przebiegu na filtr RC, o odpowiednio dużej stałej czasowej, uzyskuje się napięcie stałe zależne od wypełnienia przebiegu.
Tę metodę sterowania napięciem stałym demonstruje przykład 4.

```
;************* Ustawienie 8255 ***************
; PORT A -> PA0 – wyjście na filtr RC

PA_M EQU   0                    ; TRYB       0..2
PA_D EQU   0                    ; OUT->      0, IN->1
PCA_D      EQU   0              ; OUT->      0, IN->1
PB_M EQU   0                    ; TRYB 0..1
PCB_D      EQU   0              ; OUT->      0, IN->1

PA     EQU   PA_M*4+PA_D*2+PCA_D
PB     EQU   PB_M*4+PB_D*2+PCB_D
SET_8255     EQU   80h+PA*8+PB

;************** Ustawienie TIMERów ************

;TIMER 0
T0_G  EQU   0                   ; GATE
T0_C  EQU   0                   ; COUNTER/-TIMER
T0_M  EQU   1                   ; MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8
; TIMER 1
T1_G  EQU   0                   ;GATE
T1_C  EQU   0                   ; COUNTER/-TIMER
T1_M  EQU   0                   ; MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

;*******************************************

        LJMP  START

;************ Przerwanie Timer 0 *****************
        ORG   0BH
        CPL   F0                        ; flaga stanu wyjścia
        PUSH  PSW
        PUSH  ACC
        JB    F0, OUT_1

        MOV   TH0, R6                   ; okres 0 na wyjściu
        MOV   A, #0
        MOVX        @R1, A
```

```
        POP   ACC
        POP   PSW
        RETI

OUT_1:                                  ; okres 1 na wyjściu
        MOV   TH0, R7
        MOV   A, #1
        MOVX          @R1, A
        POP   ACC
        POP   PSW
        RETI

;*********************************************
        ORG   100H
START:
        MOV   R1, #CS55D
        MOV   A, #SET_8255
        MOVX          @R1, A
        MOV   R1, #CS55A

        LCALL         LCD_CLR
        MOV   TMOD, #TMOD_SET           ; Timer 0 liczy czas
        SETB  EA                        ; włącz  zezwolenie na
        SETB  ET0                       ; przerwanie od Timera 0

        SJMP  STOP_0                    ; na początek 0 na wyjście

LOOP:
        LCALL         WAIT_KEY          ; numer klawisz 0..15
        MOV   R2, A                     ; jako wypełnienie
        LCALL         LCD_CLR
        MOV   A, R2
        LCALL         WRITE_HEX         ; wpisz na LCD
        MOV   A, R2

        JZ    STOP_0                    ; dla 0 i 15 zatrzymaj
        CPL   A                         ; timery, na wyjście
        INC   A                         ; odpowiednio stan 0 lub 1
        MOV   R7, A                     ; dla 1...14 wpisz:
        ADD   A, #0FH                   ; do R7 okres stanu 1
        JZ    STOP_1                    ; a do R6 okres stanu 0
CPL     A
        INC   A
        MOV   R6, A
        SETB  TR0                       ; start Timera 0
        SJMP  LOOP                      ; 0 na wyjście


STOP_0:
```

```
        CLR   TR0                    ; stop Timera 0
        MOV   A, #0                  ; 1 na wyjście
        MOVX        #R1, A
        SJMP  LOOP

STOP_1:
        CLR   TR0
        MOV   A, #1
        MOVX        @R1, A
        SJMP  LOOP
```

Program ten wytwarza na linii PA0 układu 8255 (złącze wejść/wyjść cyfrowych) przebieg prostokątny o wypełnieniu ustalonym z klawiatury systemu DSM-51. Naciśnięcie klawisza o kodzie n powoduje ustawienie wypełnienia przebiegu wyjściowego na n/15. ; wpisz na LCD

Po podłączeniu do wyjścia PA0 filtru RC przedstawionego na rysunku, napięcie na kondensatorze przyjmuje wartości od 0 do 5V co 1/3V (przy założeniu, że napięcie na wyjściu PA0 przyjmuje dokładnie wartości 0 i 5V).
Wypełnienie przebiegu wyjściowego można zmieniać z dużo większą dokładnością uzyskując praktycznie liniową regulację napięcia na wyjściu filtru RC.

Wadą takiej metody wytwarzania napięcia o regulowanej wartości jest konieczność ciągłej kontroli wypełnienia generowanego sygnału. Niektóre mikrokontrolery są wyposażone w timery, które mogą całkowicie przejąć to zadanie. Mikrokontroler musi zająć się takim timerem tylko wtedy, gdy trzeba ustawić nowe parametry (okres, wypełnienie) generowanego przebiegu.
Pomiar wielkości analogowych może być również wykonany bez zastosowania scalonego przetwornika A/C. Badana wielkość jest najczęściej zamieniona na czas, który następnie jest mierzony przez mikrokontroler.
Taką metodę zastosowano do pomiaru rezystancji czujników temperatury w modelu M-10 (Miernik i regulator temperatury).

W modelu zastosowano dwa czujniki KTY10. Są to rezystancyjne czujniki temperatury. Ich rezystancja w temperaturze 25°C wynosi 2kΩ (± 1 %). Zależność rezystancji czujnika od temperatury wyraża się wzorem:

$$R_T = R_{25} * (1 + \alpha * \Delta T_A + \beta * \Delta T_A^2) = f(T_A)\ [\Omega]$$

$$\alpha = 7.88 * 10^{-3}\ [K^{-1}]$$

$$\beta = 1.937 * 10^{-5}\ [K^{-2}]$$

System mierzy wartość rezystancji czujników poprzez porównanie czasów ładowania kondensatora przez czujnik i przez rezystor wzorcowy 2kΩ ± 1 % lub jeden z czujników temperatury. Gdy napięcie na kondensatorze osiągnie próg przełączania wzmacniacza z wejściem Schmitta, stan na jego wyjściu zmieni się z 0 na 1. Ta zmiana stanu odczytana z wejścia PC0 złącza wejść/wyjść cyfrowych jest sygnałem zakończenia pomiaru czasu. Przed rozpoczęciem kolejnego procesu ładowania kondensatora jest on ponownie

rozładowywany przez klucz K3. Poszczególne klucze są zamykane, kiedy na odpowiedniej linii sterującej panuje stan 0.
Można wykazać, że mimo iż proces ładowania kondensatora przez rezystor przebiega wykładniczo, to czas ładowania tego samego kondensatora z tego samego źródła napięcia, aż do osiągnięcia tej samej progowej wartości napięcia jest wprost proporcjonalny do rezystancji rezystora, przez który następuje ładowanie. Opierając się na tej zależności można wyliczyć wartość mierzonego rezystora na podstawie pomierzonych czasów ładowania i znajomości wartości rezystora wzorcowego.
Zaletą tej metody pomiaru rezystancji przez mikrokontroler jest to, że jedyną wielkością wzorcową jest wartość rezystora. Pozostaje wielkości (napięcie VCC, pojemność kondensatora C1, próg przełączania układu Schmitta, częstotliwość zegara mikrokontrolera) nie wpływają na wynik pomiaru, jeśli tylko mają stabilne wartości.
Zastosowane w układzie klucze muszą mieć odpowiednio niską rezystancję, aby nie stały się źródłem błędów.
Przykład 5 zawiera prosty program wykorzystujący tę metodę pomiaru.

```
;*************** Ustawienie 8255 ****************

; PORT A
; A0 =0 -> pomiar rezystora wzorcowego R1=2k 1%
; A1 =0 -> pomiar czujnika temperatury R2- KTY10-6
; A2 =0 -> pomiar czujnika temperatury R3- KTY10-6
; A3 =0 -> rozładowanie kondensatora pomiarowego
; PORT C
; C0 = 0->1 – koniec pomiaru = przerwanie IPB
;C4 =1-> podgrzewanie czujnika temperatury R3
; C5 –0 -> blokada przerwania IPA

PA_M EQU   0                    ; TRYB      0..2
PA_D  EQU  0                    ; OUT->     0, IN->1
PCA_D      EQU   0              ; OUT->     0, IN->1
PB_M EQU   0                    ; TRYB 0..1
PCB_D      EQU   0              ; OUT->     0, IN->1

PA     EQU   PA_M*4+PA_D*2+PCA_D
PB     EQU   PB_M*4+PB_D*2+PCB_D
SET_8255     EQU   80H+PA*8+PB

;************* Ustawienie TIMERów ************

;TIMER 0 – czas ładowania kondensatora
T0_G  EQU   0                   ; GATE
T0_C  EQU   0                   ; COUNTER/-TIMER
T0_M  EQU   1                   ; MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8
; TIMER 1
T1_G  EQU   0                   ;GATE
T1_C  EQU   0                   ; COUNTER/-TIMER
T1_M  EQU   0                   ; MODE (0..3)
```

```
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

;*****************************************

        LJMP  START
        ORG   0100H
START:
        MOV   R0, #CS55D            ; ustawienie układu 8255
        MOV   R1, #CS55C
        MOV   A, #SET_8255
        MOVX        @R0, A
        CLR   A                     ; wyłączenie grzałki
        MOVX        @R1, A

        MOV   TMOD, #TMOD_SET       ; Timer 0 – 16bit
        MOV   R0, #CS55A            ; adres kluczy pomiarowych

;*****************************************
LOOP:
        MOV   A, #07H               ; rozładowanie
        MOVX        @R0, A          ; kondensatora
        MOV   A, #10
        LCALL       DELAY_MS
        MOV   A, #0FH               ; wyłączenie kluczy
        MOVX        @R0, A

        MOV   TL0, #0               ; zerowanie Timera 0
        MOV   TH0, #0

        MOV   A, #0DH               ; pomiar rezystora R2
        MOVX        @R0, A          ; rozpoczęcie pomiaru
        SETB  TR0                   ; strat licznika

POMIAR_LOOP:                        ; oczekiwanie na
        MOVX        A, @R1          ; naładowanie kondensatora
        JNB   ACC.0, POMIAR_LOOP

        CLR   TR0                   ; kondensator naładowany
        MOV   A, #0FH               ; - koniec pomiaru
        MOVX        @R0, A          ; wyłączenie kluczy
        LCALL       LCD_CLR
        MOV   A, TH0
        LCALL       WRITE_HEX
        MOV   A, TL0
        LCALL       WRITE_HEX

        MOV   A, #5                 ; pomiary co 0.5 sek
```

```
LCALL      DELAY_100MS
SJMP  LOOP
```

W przykładzie tym do pomiaru czasu zastosowano Timer 0 pracujący w trybie 16-bitowym. Timer ten liczy czas od włączenia ładowania kondensatora przez rezystor R2 do momentu naładowania kondensatora, tj. stanu 1 na wejściu PC0. Wynik pomiaru powtarzanego co 0.5 s wyświetlany jest na wyświetlaczu LCD. Podgrzewanie rezystora R2 ręką daje się od razu zauważyć jako wzrost mierzonej wartości.
Wykonanie dodatkowo pomiaru rezystora wzorcowego pozwala na obliczenie dokładnej wartości mierzonego rezystora, a następnie temperatury.

ZADANIA

ZADANIE 1

Korzystając z przykładu 2 zdjąć obie charakterystyki diody Zenera o napięciu Zenera mniejszym od 4V.

ZADANIE 2

Dlaczego w modelu M-02 mierzy się napięcie na wejściu IN0, czyli napięcie z wyjścia analogowego OUT? Korzystając z przykładu 3 zdjąć charakterystyki tranzystora dla różnych prądów bazy.

ZADANIE 3

Sprawdzić, czy napięcie uzyskiwane w przykładzie 4 jest liniowo zależne od wypełnienia przebiegu. Zmodyfikować przykład tak, aby ustalać napięcie wyjściowe z rozdzielczością 8 bitów.

ZADANIE 4

Zmodyfikować przykład 5 tak, aby mierzyć rezystor wzorcowy R1 i rezystor R3. Obliczyć temperaturę otoczenia. Do jakiego temperatury można podgrzać rezystor R3 za pomocą grzałki T1?

WSKAZÓWKI

Ad. 2

Przetworniki C/A i A/C nie są ze sobą kalibrowane. Dlatego od obliczenia prądu lepiej jest przyjąć wartości napięć na obu końcach rezystora zmierzone przez przetwornik A/C.
Dodatkowo wyjście z przetwornika C/A ma ograniczoną wydajność prądową. Przy pomiarach charakterystyk tranzystorów dla dużych prądów bazy można zaobserwować rozbieżność pomiędzy teoretyczną wartością napięcia na wyjściu OUT a wartością zmierzoną na wejściu IN0.

Ad. 3

Modyfikując przykład 4 należy pamiętać, iż okres przebiegu nie powinien się znacznie wydłużyć, gdyż napięcie na filtrze RC miałoby duże tętnienia. Należy zatem modyfikować zarówno wartość TH0 jak TL0, co jest dosyć kłopotliwe – trzeba uwzględniać naliczoną już od momentu przepełnienia timera wartość w TL0. Lepszym rozwiązaniem jest wybranie innego trybu pracy timera. Którego?

Ad. 4

Warunki obu pomiarów powinny być możliwie do siebie zbliżone. Szczególnie istotne jest, aby zadbać o jednakowe rozładowanie kondensatora, stosując do tego ten sam czas rozładowania. W przykładzie umożliwić włączenie grzałki wybranym klawiszem.

ĆWICZENIE 9

KLAWIATURY MATRYCOWE. KLAWIATURY SEKWENCYJNE.

Wstęp

Jednym z podstawowych środków komunikacji systemu mikroprocesorowego z użytkownikiem jest klawiatura. Najbardziej rozpowszechnione sposoby podłączenia klawiatury do mikrokontrolera polegają na wykorzystaniu w tym celu portów. Porty jako układy we/wy zapewniają komunikację dwukierunkową. W mikrokontrolerze do odczytu danej z portu wykorzystuje się możliwość zmiany stanu z 1 na 0 poprzez sygnał zewnętrzny. Jak z tego wynika porty służące do odczytu muszą być każdorazowo ustawione w stan 1. Ograniczona liczba linii w porcie nie pozwala na najprostsze rozwiązanie tzn. przypisanie jednego klawisza klawiatury do jednej linii w porcie. W celu powiększenia rozmiaru klawiatury stosuje się dwa popularne rozwiązania:
- klawiaturę sekwencyjną - poprzez zewnętrzny bufor podłączony do szyny mikrokontrolera sterowana jest jedna linia w porcie. W systemie DSM51 zorganizowana jest klawiatura złożona z sześciu przycisków [↵], [Esc], [←], [↑], [→], [↓]. Do obsługi wybrano linię P3.5 sterowaną buforem CSDS.
- klawiaturę matrycową - w tego typu klawiaturze następuję podział na kolumny i wiersze zapewniające poszczególnym klawiszom indywidualne współrzędne po których można je rozpoznać . W systemie DSM51 klawiatura matrycowa podłączona jest za pomocą portów zewnętrznych oraz linii adresowych A0 i A1. Rolę kolumn spełniają tutaj linie adresowe A0 i A1 natomiast wierszy linie danego portu zewnętrznego. Porty zewnętrzne znajdują się pod adresami 21H oraz 22H i zostały oznaczone jako CSKB0 i CSKB1. Klawiatura matrycowa składa się z 16 klawiszy: 0, 1, 2, 3, 4, 5, 6, 7 podłączonych do portu CSKBO (adres 21H) oraz 8, 9, [←], [↑], [→], [↓], Esc, Enter [↵] podłączonych do portu CSKB1 (adres 22H).
Schematy elektryczne klawiatur zamieszczono w dodatku do tej instrukcji

W ćwiczeniu należy zapoznać się z zasadami korzystania z określonego typu klawiatury oraz realizacjami programowymi nad odczytem wybranych klawiszy.

ZADANIE 1

W zadaniu 1 należy zaobserwować sposób odczytu danej pojawiającej się w momencie naciśnięcia uaktywnionego klawisza. Należy rozpoznać adres bufora CSDS sterującego klawiaturą sekwencyjną.

```
LED    EQU   P1.7
KEY    EQU   P3.5                    ;odczyt klawiatury

;Stałe używane w programie
KEY_COD EQU      00001001B           ;wybrane klawisze – 1 i 4

       LJMP  START
```

```
        ORG   100H
START:

        MOV   R0,#CSDS          ;adres wyboru klawisza
        MOV   A,#KEY_COD
        MOVX        @R0,A       ;wpisz wybrane klawisze

LOOP:
        MOV   C,KEY             ;czytaj stan klawiszy
        CPL   C                 ;neguj
        MOV   LED,C             ;zapal diodę TEST gdy
                                ;klawisz jest naciśnięty

        SJMP  LOOP
```

Wykonaj program w trybie krokowym. Zmodyfikuj podany przykład tak by uaktywnić inne klawisze klawiatury sekwencyjnej.

ZADANIE 2

W zadaniu drugim zaprezentowano program eliminujący drgania styków klawiatury mogące wprowadzać błędy w odczycie. Zastosowano tu pętlę opóźnienia czasowego, w trakcie której powinien ustabilizować się stan linii P3.5 Należy porównać

```
LED   EQU   P1.7
KEY   EQU   P3.5                ;odczyt klawiatury

;Stałe używane w programie
KEY_COD EQU       00001001B     ;wybrane klawisze-1 i 4

        LJMP  START
        ORG   100H
START:

        MOV   R0,#CSDS          ;adres wyboru klawisza
        MOV   A,#KEY_COD
        MOVX        @R0,A       ;wpisz wybrane klawisze

LOOP_NO:
        JNB   KEY,LOOP_NO       ;czekaj na naciśnięcie
                                ;klawisza
        MOV   A,#10             ;eliminacja drgań styków
        LCALL       DELAY_MS
        JNB   KEY,LOOP_NO       ;czy nadal naciśnięty

        CPL   LED               ;neguj stan diody TEST
                                ;przy naciśnięciu klaw.
```

```
LOOP_YES:
        JB      KEY,LOOP_YES            ;czekaj na zwolnienie
                                        ;klawisza
        MOV     A,#10                   ;eliminacja drgań styków
        LCALL           DELAY_MS
        JB      KEY,LOOP_YES            ;czy klawisz zwolniony

        SJMP    LOOP_NO                 ;powrót do pętli czekania
                                        ;na klawisz
```

ZADANIE 3

Do samodzielnego wykonania.

Zmodyfikuj przykład z zadania 2 tak by w pętli programowej kolejno przeglądać w interwale czasowym 20 ms kolejne stany klawiszy. Należy rozpoznawać numer naciśniętego klawisza a następnie wyświetlać go na wyświetlaczu LCD korzystając z odpowiednich podprogramów.

ZADANIE 4

Podany przykład zawiera trzy pętle programowe: LOOP_NO, LOOP_NR oraz LOOP_YES. Pierwsza pętla służy do ciągłego odczytu danych z portu CSKB0. W momencie naciśnięcia jednego z przycisków 0 - 7 poprzez wykorzystanie instrukcji JZ (skocz jeśli wartość akumulatora =0) następuje przejście do pętli następnej - LOOP_NR. Pętla ta służy do rozpoznania numeru naciśniętego klawisza. Należy zwrócić uwagę na instrukcję RRC A. Instrukcja ta służy do obrotu w prawo zawartości akumulatora poprzez bit C. W trybie pracy krokowej należy zaobserwować i zanotować kolejne wartości akumulatora w trakcie wykonywania instrukcji RRC A dla kilku różnych przycisków. W chwili zdekodowania numeru przycisku (gdy bit C =0) i wyświetleniu informacji o tym na wyświetlaczu LCD program przechodzi do trzeciej pętli LOOP_YES. W pętli tej rozpoznawane jest czy nadal pozostaje naciśnięty klawisz klawiatury. Jeśli wszystkie przyciski zostaną zwolnione program powraca do pętli LOOP_NO.

```
        LJMP            START
        ORG             100H
START:

        LCALL                   LCD_CLR
        MOV             R0,#CSKB0               ;adres klawiszy 0..7

LOOP_NO:
        MOVX                    A,@R0           ;odczyt stanu klawiszy
        CPL             A
        JZ              LOOP_NO                 ;czy klawisz naciśnięty

        MOV             R2,#0FFH
```

```
LOOP_NR:                                ;przekoduj nr klawisza
                                        ;kod 1z 8 na cyfrę 0..7
                                        ;pętla będzie wykonana 1-8 razy
        INC     R2                      ;więc R2 osiągnie wartość 0..7
        RRC     A                       ;obrót akumulatora przez C
                                        ;aż odczytana 1 wsunie się do C
        JNC     LOOP_NR

        MOV     A,R2                    ;przekoduj nr klawisza
        ADD     A,#30H                  ;na kody znaków wyśw. LCD
        LCALL           WRITE_DATA      ;wypisz kod klawisza

LOOP_YES:
        MOVX            A,@R0           ;odczyt stanu klawiszy
        CPL     A
        JNZ     LOOP_YES                ;czy klawisz zwolniony

        SJMP    LOOP_NO                 ;powrót do oczekiwania na
                                        ;naciśnięcie klawisza
```

Wykonaj program w trybie krokowym. Rozpoznaj adresy buforów sterujących pierwszą grupą klawiszy 0 - 7. Zmodyfikuj program dla drugiej sekcji tak by wysterować klawisze 8 - Enter.

ZADANIE 5

Zmodyfikuj program z zadania 4 tak by wyeliminować drgania styków klawiatury. Zaproponuj algorytm programu dla obsługi całej klawiatury.

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ĆWICZENIE 10

WYŚWIETLACZ LCD. WYŚWIETLACZ SEKWENCYJNY.

W lekcji 6 przedstawiony był sposób sterowania i wyświetlania znaków na wyświetlaczu 7-segmentowym. Zasadniczym ograniczeniem w tego typu wyświetlaczach jest możliwość wyświetlania tylko cyfr. Wynika to bezpośrednio z ich budowy. Oprócz zdefiniowania dodatkowo kilku liter, nie ma możliwości wypisywania na nich słownych komunikatów. Dlatego też coraz powszechniej stosowane są alfanumeryczne wyświetlacze ciekłokrystaliczne (ang. LCD – Liquid Crystal Display).
W wyświetlaczach tych każdy znak zdefiniowany jest na polu 5x7 punktów, co pozwala na wyświetlanie dowolnych znaków (cyfr, liter) w pełni zrozumiałych dla człowieka. Ze względu na znacznie większą ilość danych niż w wyświetlaczu 7-segmentowym oraz na bardziej złożone sterowanie, wyświetlacze LCD są standardowo wyposażone w specjalizowane procesory, które zarządzają wyświetlaniem.
Takie procesory nazywane są sterownikami wyświetlacza. Typowym ich przedstawicielem jest układ firmy Hitachi HD 44780. Do takiego sterownika mikrokontroler wysyła tylko dane (które mają być wyświetlane) i instrukcje (w jaki sposób mają być wyświetlane). Natomiast sposób zamiany danych na punkty, które mają świecić, czy przebiegi sterujące wyświetlaniem, to już zadanie sterownika.
Sterownik HD 44780 jest przystosowany do innych sygnałów sterujących niż te, które występują w systemach opartych na mikrokontrolerze 8051. Dzięki zbudowaniu dekodera adresów na układzie typu GAL, udało się wytworzyć specjalne dla wyświetlacza LCD inne sygnały sterujące, niż dla pozostałych elementów systemu.
Wyświetlacz LCD jest tak podłączony, że zajmuje w przestrzeni adresowej cztery kolejne komórki pamięci, począwszy od adresu 80_H ($FF80_H$). Każdy z tych adresów pełni specyficzną rolę:

80$_H$ – zapis instrukcji
81$_H$ – zapis danych,
82$_H$ – odczyt stanu,
83$_H$ - odczyt danych.

Po wysłaniu do sterownika wyświetlacza LCD kolejnej instrukcji bądź kolejnych danych, sterownik musi wykonać otrzymane polecenie, tzn. wykonać instrukcję bądź umieścić dane pod odpowiednim adresem. Na wykonanie tych operacji sterownik potrzebuje określonego czasu. W tym czasie sterownik jest zajęty i nie przyjmuje kolejnych poleceń. Jedynym wyjątkiem jest możliwość odczytania stanu.
Przed wydaniem kolejnego polecenia należy sprawdzić, czy sterownik jest gotów do jego przyjęcia. Jest to możliwe przez odczytanie stanu wyświetlacza, czyli odczyt spod adresu 82H. Siódmy bit stanu jest to flaga Busy (flaga zajętości). Jeżeli flaga ta jest równa 1, to sterownik jest zajęty i będzie głuchy na nasze polecenia. Jeżeli flaga Busy równa się 0, to można wysłać do sterownika polecenie.
Przykład 1 ilustruje sposób wprowadzania kolejnych danych na wyświetlacz.

```
        LJMP    START
        ORG     100H
START:
```

```
        LCALL   LCD_CLR
        MOV     R0,#LCDWD       ;adres wpisywania danych
                                ;na wyświetlacz LCD
        MOV     R1,#LCDRC       ;adres odczytu stanu
                                ;wyświetlacza LCD
LOOP:
        LCALL   WAIT_KEY        ;pobierz klawisz
        ADD     A,#30H          ;zamiana kodu klawisza
        MOV     R2,A            ;na kod znaku LCD
BUSY:
        MOVX    A,@R1           ;odczyt stanu
        JB      ACC.7,BUSY      ;oczekiwanie na BUSY=0
        MOV     A,R2            ;wysłanie kodu znaku
        MOVX    @R0,A           ;do wyświetlacza LCD

        SJMP    LOOP
```

Każdy rozpoznany przez podprogram WAIT_KEY klawisz jest po przekodowaniu wysyłany na wyświetlacz. Przed wysłaniem kolejnych danych procesor sprawdza, czy sterownik wyświetlacza jest gotów na przyjęcie znaku – pętla BUSY. W pętli tej procesor czyta stan wyświetlacza , aż do momentu, kiedy flaga Busy będzie równa 0. Wtedy wysyłany jest kolejny znak. Wyświetlane znaki zależą od sposobu przekodowywania klawiszy na znaki. Zmieniając przekodowywanie w prosty sposób można przyporządkować klawiszom na przykład kolejne litery.
Po wprowadzeniu 16-tu znaków kursor znika z wyświetlacza i kolejne znaki pozornie nie są wpisywane na wyświetlacz. Jednak gdy wprowadzonych zostanie ponad 40 znaków, kursor ponownie pojawia się na wyświetlaczu – tym razem w dolnej linii. To „dziwne” zjawisko wynika z tego, że wyświetlacz LCD wyposażony jest w uniwersalny sterownik LCD. Jego uniwersalność polega na tym, że bez względu na to, jakim wyświetlaczem steruje (2 linie po 16 znaków, 2x20, 2x40, 1x80), jest to zawsze ten sam sterownik, który pamięta 80 znaków (1 linia x 80 znaków bądź 2 linie x 40 znaków). Tak więc wysłane na wyświetlacz znaki nie zginęły, a jedynie nie można było ich wszystkich naraz wyświetlić.
Zjawisko to nie występuje przy wpisywaniu znaków na wyświetlacz LCD za pomocą standardowych podprogramów zawartych w pamięci EPROM. W podprogramy te został wprowadzony mechanizm wykrywania końca linii wyświetlacza 2x16 i automatycznego przenoszenia kursora na początek drugiej linii. Dzięki temu wszystkie znaki wpisane na wyświetlacz są widoczne.
Przykład 2 ilustruje sposób wysyłania instrukcji sterujących pracą wyświetlacza (wyraz wszystkich instrukcji można znaleźć w dodatku F).

```
        LJMP  START
        ORG   100H
START:
        MOV   R0,#LCDWC         ; adres wpisu instrukcji
        MOV   R1,#LCDRC         ; adres odczytu stanu

        MOV   A,#1              ; kasuj dane wyświetlacza
        ACALL     WRITE
```

```
        MOV  A,#0FH             ; włącz wyświetlacz, kursor
        ACALL      WRITE        ; i mruganie kursora

        MOV  A,#06H                  ; ustaw kierunek
        ACALL      WRITE             ; przesuwania się kursora

        INC  R0                      ; adres wpisu danych
        MOV  DPTR, #TEXT             ; adres tekstu

WRITE_TXT:
        CLR  A                       ; pobranie kolejnego
        MOVC       A,@A+DPTR         ; znaku tekstu
        JZ   TEXT_END                ; bajt=0 – koniec tekstu

        ACALL      WRITE             ; wpisanie na wyświetlacz
        INC  DPTR                    ; modyfikacja adresu
                                     ; pobrania kolejnego znaku
        SJMP WRITE_TXT               ; pobierz kolejny znak

TEXT_END:
        DEC  R0                      ; adres wpisu instrukcji
        MOV  DPTR,#KEY_COD           ; adres tabeli kodowania
                                     ; klawiszy
LOOP:                                ; pętla reakcji na klawisze
        LCALL      WAIT_KEY          ; pobierz klawisz
        CJNE A,#ODH,NO_DOWN          ; czy klawisz 'v'
DOWN:                                ; klawisz 'v' ( w dół )
        MOVX       A,@R1
        JB   ACC.7,DOWN              ; oczekiwanie na BUSY=0
        MOVX       A,@R1             ; odczytanie adresu  z LCD
        CPL  ACC.6                   ; zmiana linii 1<->2
        SETB ACC.7                   ; znacznik rozkazu
        ACALL      WRITE             ; ustaw nowy adres
        SJMP LOOP

NO_DOWN:
        MOV  R2,A                    ; zapamięta klawisz
        MOVC       A,@A+DPTR         ; przekoduj klawisze
                                     ; na instrukcje
        JZ   WRITE_DAT               ; 0-klawisz jako dane

        ACALL      WRITE             ; wysłanie instrukcji
        SJMP LOOP

WRITE_DAT:                           ; wpisz znak na LCD
        MOV  A,R2                    ; odtwórz klawisz
        ADD  A,#30H                  ; modyfikuj jako znak
        INC  R0                      ; adres wpisu danych
        ACALL      WRITE             ; wpisanie znaku na LCD
```

```
        DEC   R0                        ; adres wpisu instrukcji
        SJMP  LOOP
                        ;*******************************
                        ;Podprogram wpisu danych lub instrukcji
                        ;na wyświetlacz LCD
                        ;Zakłada prawidłowe adresy w R0 i R1

WRITE:
        MOV   R2,A                      ; przechowanie danych
BUSY:
        MOVX        A,@R1               ; odczyt stanu
        JB    ACC.7, BUSY               ; oczekiwanie na BUSY=0
        MOV   A,R2                      ; odtworzenie danych
        MOVX        @R0,A               ; wysłanie danych
        RET

                        ;*******************************
                        ;Tabela przekodowania numeru klawisza
                        ;na instrukcję, 0 -> klawisz jako znak]

KEY_COD:
        DB    0,0,0         ;0,1,2
        DB    0,18H,0       ;3,4,5
        DB    1CH,0,0       ;6,7,8
        DB    0,10H,14H     ;9,<,>
        DB    02H,0,06H     ;^,v,Esc
        DB    07H           ;Enter
;*************************************

TEXT:
        DB    ` „MicroMade“       Systemy`
        DB    ‘Mikroprocesorowe’
        DB    ‘ul. Sikorskiego 33’
        DB    ’64-920 PILA’ ,0
```

Każda instrukcja do wyświetlacza LCD jest wysyłana za pomocą podprogramu WRITE. W tym podprogramie umieszczone jest oczekiwanie na zwolnienie flagi Busy. Dopiero po jej zwolnieniu następuje wysłanie instrukcji do sterownika wyświetlacza. Dzięki temu podprogram ten może być wywoływany dowolnie często, bez obaw, że któraś instrukcja nie dotrze do sterownika.
Podprogram WRITE jest uniwersalny – może wysyłać do wyświetlacza LCD zarówno instrukcje, jak i dane. Jest to uzależnione od adresu wpisanego do rejestru R0. Podprogram WRITE nie modyfikuje tego rejestru. O jego właściwą zawartość należy zadbać w programie głównym. Należy zauważyć, że również rejestr R1 musi mieć właściwą wartość do sprawdzenia stanu flagi Busy. W przeciwnym razie, podprogram WRITE nie będzie prawidłowo działał.
Na początku programu zostają wysłane do wyświetlacza i trzy instrukcje (zamiast podprogramu LCD_CLR dostępnego w pamięci EPROM):

- instrukcja 01_H kasuje dane wyświetlacza i ustawia kursor pod adresem 0,

- instrukcja $0F_H$ włącza wyświetlacz, kursor i mruganie tego kursora,
- instrukcja 06_H ustawia sposób przemieszczania kursora przy wpisywaniu danych.

Po tych wstępnych instrukcjach na wyświetlacz zostaje wysłany tekst, wpisany w programie od etykiety TEXT. Każda kolejna wartość pobierana z programu jest wysyłana do sterownika wyświetlacza za pośrednictwem podprogramu WRITE. Na początku programu do rejestru R0 wprowadzono adres służący do wysyłania instrukcji do sterownika wyświetlacza. Na okres wypisywania na wyświetlaczu tekstu (pętla WRITE_TEXT) rejestr ten jest zwiększany o 1, a więc zawiera adres do wpisywania danych.

W głównej pętli programu LOOP program oczekuje na naciśnięcie klawiszy, jednak ich kody nie są bezpośrednio wysyłane na wyświetlacz. Program rozpoznaje klawisze i podejmuje różnorodne działanie.

Najpierw wyróżniony jest klawisz [↓], którego obsługa jest inna od obsługi pozostałych (opis poniżej).

Pozostałe klawisze przekodowywane są za pomocą tabeli KEY_COD, która jednocześnie dzieli je na dwie grupy. Jeżeli dla danego klawisza z tabeli zostanie pobrana wartość różna od zera, oznacza to, że wartość ta ma być wysłana do wyświetlacza LCD jako instrukcja. Jeżeli dla danego klawisza w tabeli jest wartość 0, to należy odtworzyć numer klawisza (zapamiętany w tym celu w rejestrze R2) i przekodować go w sposób właściwy dla danych. Może to być zamiana na cyfry (jak w przykładzie) lub na litery, albo w całkiem inny, dowolny sposób, na przykład za pomocą drugiej tabeli kodującej.

Klawiszom zostały przyporządkowane różnorodne funkcje sterujące w celu przedstawienia bogatych możliwości wyświetlaczy alfanumerycznych LCD. Klawisze [4] i [6] powodują obrót danych na wyświetlaczu o jedną pozycję, odpowiednio w lewo i prawo. Dzięki temu można obejrzeć cały napis wpisany na wyświetlacz.

Klawisze [←] i [→] powodują przesunięcie pozycji kursora w lewo lub w prawo w pamięci wyświetlacza (zarówno w części widocznej jak i niewidocznej). Klawisz [↑] powoduje ustawienie danych na wyświetlaczu w pozycji wyjściowej i ustawienie kursora pod adresem 0.

Po uruchomieniu programu należy zwrócić uwagę, że obracanie danych na wyświetlaczu odbywa się jednocześnie dla obu linii, ale dane w liniach są od siebie niezależne. Natomiast kursor przesuwa się z końca jednej linii na początek drugiej i odwrotnie. Dokładne zrozumienie tych zależności wymaga przedstawienia sposobów adresowania poszczególnych pozycji wyświetlacza.

Pierwsza linia wyświetlacza zawiera adresy od 00_H...27_H (40 bajtów), a druga od 40_H...67_H. W czasie obrotu bajty są przemieszczane tylko wewnątrz adresów jednej linii. Natomiast kursor poruszany w sposób standardowy przemieszcza się po kolejnych adresach, aż napotka adres 27_H bądź 67_H. W tym przypadku jest on automatycznie przestawiany na początek drugiej linii, czyli odpowiednio pod adres 40_H lub 00_H.

Adresy odpowiadających sobie pozycji w liniach różnią się między sobą o 40_H. Bieżący adres jest odczytywany razem z flagą Busy (bity 0...6). Jednak jest on prawidłowy tylko wówczas, gdy flaga Busy równa się 0. Istnieje też specjalna instrukcja do ustawiania bieżącego adresu, czyli pozycji kursora. Te dwie możliwości zostały wykorzystane do przemieszczania kursora z górnej linii na dolną i odwrotnie. Wystarczy tylko odczytać adres, zmienić wartość bitu 6 w adresie na przeciwną i ustawić ten adres jako bieżący. Taka właśnie procedura jest wykonywana dla klawisza [↓].

Oprócz możliwości sterowania obracaniem danych na wyświetlaczu i przesuwaniem kursora, istnieją jeszcze różne sposoby wprowadzania danych na wyświetlacz. Przesłany do

wyświetlacza znak zawsze wpisywany jest pod bieżący adres wskazywany przez kursor, bez względu na to, czy adres ten jest aktualnie widoczny, czy nie.
W powyższym przykładzie klawisze [Enter] i [Esc] zostały użyte do zmiany sposobu wprowadzania danych na wyświetlacz. Po klawiszu [Enter] wprowadzenie kolejnych danych na wyświetlacz powoduje jednoczesny obrót danych na wyświetlaczu. W ten sposób kursor zostaje cały czas w jednej pozycji na wyświetlaczu i nigdy nie znajdzie się poza widocznym obszarem.
Klawisz [Esc] wyłącza ten tryb. Po wprowadzeniu kolejnych danych kursor przesuwa się o jedną pozycję na wyświetlaczu i gdy dojdzie do jego krawędzi, to przy następnym znaku, zniknie z widocznej części wyświetlacza.
Stosując te wszystkie instrukcje można napisać program wypisywania znaków na wyświetlacz zgodnie z własnymi potrzebami i upodobaniami. Sterownik HD 44780 ma jeszcze jedną bardzo użyteczną cechę. Oprócz wykorzystania standardowej tabeli znaków, można dodatkowo zdefiniować osiem znaków, na przykład polskich liter. Wprawdzie liter polskich jest więcej (9 małych i 9 dużych), ale rzadko się zdarza, aby wszystkie były naraz potrzebne.
Definiowanie własnych znaków demonstruje przykład 3.

```
        LJMP  START
        ORG   100H
START:
        MOV   R0, #LCDWC            ; adres wpisu instrukcji
        MOV   R1, #LCDRC            ; adres odczytu stanu

        MOV   A, #48H               ; ustaw adres generatora
        ACALL       WRITE           ; znaków dla znaku 1

        INC   R0                    ; adres wpisu danych
        MOV   DPTR, #LITERA         ; adres definicji litery
        MOV   R3, #8                ; licznik bajtów definicji

LOOP:                               ; wpisz definicję litery
                                    ; do generatora znaków LCD
        CLR   A
        MOVC        A, @A+DPTR      ; odczyt kolejnego bajtu
        LCALL       WRITE           ; zapis do generatora znaków
        INC   DPTR                  ; modyfikacja adresu
        DJNZ  R3,LOOP               ; przepisanie 8 bajtów

        DEC   R0                    ; adres wpisu instrukcji
        MOV   A, #1                 ; kasuj dane wyświetlacza
        ACALL       WRITE

        MOV   A, #0FH               ; włącz wyświetlacz, kursor
        ACALL       WRITE           ; i mruganie kursora

        MOV   A, #06H               ; ustaw kierunek przesuwu
        ACALL       WRITE           ; kursora
```

```
        INC     R0                      ; adres wpisu danych
        MOV     DPTR, #TEXT             ; adres tekstu do
                                        ; wyświetlenia na LCD

WRITE_TXT:                              ; wpisz tekst na LCD
        CLR     A
        MOVC            A, @A+DPTR      ; pobranie znaku tekstu
        JZ      TEXT_END                ; bajt = 0 – koniec tekstu
        ACALL           WRITE           ; wpis na wyświetlacz
        INC     DPTR                    ; modyfikacja adresu
        SJMP    WRITE_TXT               ; wpisz kolejny znak

TEXT_END:
        SJPM    $                       ; koniec programu

;*******************************
;Podprogram wpisu danych lub instrukcji
; na wyświetlaczu
; Zakłada prawidłowe adresy w R0 i R1

WRITE:
        MOV     R2,A                    ; przechowanie danych
BUSY:
        MOVX            A, @R1          ; odczytanie stanu
        JB      ACC.7,BUSY              ; oczekiwanie na BUSY = 0
        MOV     A,R2                    ; odtworzenie danych
        MOVX            @R0,A           ; wysłanie danych
        RET

;***************************
; Tabela batów definiująca literę ‘ń’

LITERA:
        DB      00000010B
        DB      00000100B
        DB      00010110B
        DB      00011001B
        DB      00010001B
        DB      00010001B
        DB      00010001B
        DB      00000000B

;***************************
TEXT:
        DB      ‘Gdansk
        DB      ‘Gda’,1,’sk’,0
```

W przykładzie tym została zdefiniowana jedna polska litera „ń”.
Na definicję litery składa się 8 bajtów, w których jedynki tworzą wizerunek litery. Definiując te liczby w sposób binarny, można dostrzec obraz zdefiniowanej litery. Tak zdefiniowaną literę trzeba przesłać do specjalnego obszaru pamięci wyświetlacza, zwanego generatorem znaków.
Generator znaków zawiera 64 bajty odpowiednio dla znaków od 0...7. Litera „ń” została wpisana jako znak 1, a więc od adresu 08_H...$0F_H$. W celu wpisywania znaku do generatora znaków należy ustawić odpowiedni adres. Służy do tego specjalna instrukcja. Po ustawieniu adresu przesyłane są bajty, które umieszczane są w kolejnych komórkach pamięci. Od tej pory jako znak numer 1 zostanie wyświetlona litera „ń”.
Dla porównania na wyświetlacz został wprowadzony ciąg znaków ‘Gdansk Gda’,1,’sk’, który jest widoczny jako „Gdansk Gdańsk”.

ZADANIA

ZADANIE 1

Wykorzystując poznane instrukcje sterujące, napisz program, który po wprowadzeniu długiego tekstu (na przykład z przykładu 2) będzie go automatycznie obracał tak, aby można było go w całości odczytać.

ZADANIE 2

Zdefiniuj inne polskie litery i wypisz tekst zawierający ich maksymalnie dużo.

ZADANIE 3

Napisz program wprowadzania na wyświetlacz z klawiatury liczb dziesiętnych w taki sposób, aby starsze cyfry odsuwały się w lewą stronę, a kolejne cyfry zawsze były wpisywane w tym samym miejscu.

WSKAZÓWKI

Ad. 1
Program ten po umieszczeniu tekstu na wyświetlaczu musi jedynie wydawać komendy obrotu tekstu. Regulując częstotliwość wysyłania tych komend można otrzymać różną prędkość obracania napisu.
Przykład 4 zawiera jedno z możliwych rozwiązań.

```
        LJMP  START
        ORG   100H
START:
        MOV   R0, #LCDWC          ; adres wpisu instrukcji
        MOV   R1, #LCDRC          ; adres odczytu stanu

        MOV   A, #1               ; kasuj dane wyświetlacza
        ACALL       WRITE

        MOV   A, #0FH             ; włącz wyświetlacz, kursor
        ACALL       WRITE         ; i mruganie kursora
```

```
        INC     R0                      ; adres wpisu danych
        MOV     DPTR, #TEXT             ; adres tekstu

WRITE_TXT:
        CLR     A                       ; pobranie kolejnego
        MOVC            A, @A+DPTR      ; znaku tekstu
        JZ      TEXT_END                ; bajt=0 – koniec tekstu

        ACALL           WRITE           ; wpisanie na wyświetlacz
        INC     DPTR                    ; modyfikacja adresu
                                        ; pobrania kolejnego znaku
        SJMP    WRITE_TXT               ; pobierz kolejny znak

TEXT_END:
        DEC     R0                      ; adres wpisu instrukcji

LOOP:                                   ; pętla animacji
        MOV     A,#4
        LCALL           DELAY_100MS     ; co 0.4 sek

        MOV     A,#18H                  ; obrót danych w lewo
        ACALL           WRITE           ; na wyświetlaczu LCD

        SJMP    LOOP

;*********************************
;Podprogram wpisu danych lub instrukcji
;na wyświetlacz LCD
;Zakłada prawidłowe adresy w R0 i R1

WRITE:
        MOV     R2,A                    ; przechowanie danych
BUSY:
        MOVX            A, @R1          ; odczytanie stanu
        JB      ACC.7,BUSY              ; oczekiwanie na BUSY = 0
        MOV     A,R2                    ; odtworzenie danych
        MOVX            @R0,A           ; wysłanie danych
        RET

;*************************
;TEXT:
        DB      „MicroMade” Systemy’
        DB      ‘Mikroprocesorowe    ‘
        DB      ‘ul. Sikorskiego 33    ‘
        DB      ‘       64-920 PILA   ‘,0
```

Ad. 2

Można oprzeć się na przykładzie 3. Prawidłowość zdefiniowania liter można sprawdzić obserwując je na wyświetlaczu.

Ad. 3

Można to osiągnąć poprzez odpowiednie kody wysłane do wyświetlacza przed wprowadzeniem liczby. Można to wypróbować obserwując działanie przykładu 2.

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ĆWICZENIE 11.

OPERACJE ARYTMETYCZNE

Lekcja ta przedstawia wykonanie operacji arytmetycznych na liczbach jednobajtowych, różnorodne formaty danych oraz sposoby użycia wyświetlacza LCD i klawiatury matrycowej przy korzystaniu z podprogramów systemu DSM-51.
Mikrokontroler 8051 jest mikrokontrolerem 8-bitowym. Oznacza to że podstawowa jednostka jego pamięci składa się z 8 bitów i nazywana jest bajtem. Ponieważ każdy bit może być ustawiony na 0 lub 1, bajt może pamiętać 2^8 =256 różnych stanów, zaczynając od 0000 0000 i kończąc na 1111 1111. Najczęściej tych 256 różnych stanów traktuje się jako liczbę z zakresu 0...255.
Mikrokontroler 8051 posiada wbudowaną jednostkę arytmetyczno logiczną. (ALU – Arithmetic Logic Unit), która potrafi wykonywać operacje na liczbach jednobajtowych. Można ą porównać do kalkulatora, który potrafi wykonywać podstawowe działania (dodawanie, odejmowanie, mnożenie, dzielenie), ale ego wyświetlacz zawiera tylko jedną cyfrę. Wszystkie dane do działań oraz wyniki muszą się zawierać w zakresie 0...9. Taki kalkulator potrafi liczyć, ale wykonanie działań na liczbach większych jest raczej trudne.
W niniejszej lekcji są omówione tylko proste operacje arytmetyczne na danych jednobajtowych.
W lekcji 2 przedstawione zostały różne sposoby zapisu liczb. Na przykład, dla liczby 10 wyglądało to następująco:

- Zapis dziesiętny: 10
- Zapis binarny: $0000\ 1010_B$
- Zapis szesnastkowy: $0A_H$

Dla rozróżnienia tych zapisów na końcu liczby umieszcza się literę B dla zapisu binarnego lub H dla szesnastkowego (heksadecymalnego).
Wszystkie te zapisy, mimo że wyglądają różnie, przedstawiają tą samą wartość. W mikrokontrolerze jest ona zawsze pamiętana jako 8 kolejnych bitów, czyli tak jak jest to w zapisie binarnym.
Dal przypomnienia przedstawiono liczby z zakresu 0...15 zapisane na różne sposoby.
W systemie DSM-51 wśród standardowych programów dostępny jest program WRITE_HEX, który wypisuje na wyświetlaczu LCD bajt z akumulatora w postaci szesnastkowej. Przy użyciu tego podprogramu na wyświetlacz wypisywana jest liczba, zawsze jako 2 znaki z zakresu 0...9, A,B,C,D,E,F. Podprogram ten jest wykorzystywany w przykładach do obejrzenia wyników poszczególnych działań. Należy zwrócić uwagę, że zapis szesnastkowy liczb 0...9 jest równoznaczny z zapisem dziesiętnym, a więc przy liczbach mniejszych od 10 można zapomnieć o przeliczaniu zapisu szesnastkowego na dziesiętny. Przykład 1 demonstruje dodawanie

```
        LJMP  START
        ORG   100H
START:

        LCALL      LCD_CLR    ;wyczyść wyświetlacz LCD
        MOV  A,#2             ;wpisz do akumulatora liczbę 2
        ADD  A,#2             ;dodaj do akumulatora liczbę 2, wynik w akumulatorze
        LCALL      WRITE_HEX  ;akumulator na LCD
```

```
        LJMP  $
```

Przykład ten potwierdza, że 2+2=4. Użyty tu rozkaz ADD powoduje dodanie do zawartości akumulatora wartości wskazane na drugiej pozycji. Wartość ta może być: stałą (oznaczoną przez #), zawartością rejestru (R0...R7), zawartością komórki pamięci adresowane bezpośrednio (direct) lub zawartością komórki adresowane pośrednio (@Ri).
Wynik wykonania tego rozkazu, czyli suma, jest zawsze zapisywany do akumulatora. Dlatego też bezpośrednio po rozkazie ADD można użyć podprogramu WRITE_HEX.
Rzadko się zdarza, że w trakcie pisania programu wpisuje się dane do operacji arytmetycznych w sposób jawny w programie. W powyższym przykładzie można przecież od razu napisać

```
        MOV  A,#4
```

Zamiast dodawać dwa do dwóch.
Przeważnie, 1 wartość do operacji jest wyliczana w trakcie trwania programu lub pochodzi z zewnątrz, na przykład z klawiatury.
W przykładzie 2 liczby, które mają być dodane, odczytywane są z klawiatury.

```
        LJMP  START
        ORG   100H
START:

        LCALL       LCD_CLR

        LCALL       WAIT_KEY        ; pobierz pierwszy czynnik
                                    ; z klawiatury matrycowej
        MOV  R0, A                  ; zapamiętaj w R0
        LCALL       WRITE_HEX       ; wyświetl na LCD

        MOV  A, #'+'                ; znak sumy wyświetl
        LCALL       WRITE_DATA      ; na LCD jako znak

        LCALL       WAIT_KEY        ; pobierz drugi czynnik
        MOV  R1, A                  ; zapamiętaj w R1
        LCALL       WRITE_HEX       ; wyświetl na LCD

        MOV  A, #'='                ; znak równości
        LCALL       WRITE_DATA      ; wyświetl jako znak

        MOV  A, R0                  ; pierwszy czynnik do A
        ADD  A, R1                  ; dodaj drugi czynnik
                                    ; wynik w akumulatorze
        LCALL       WRITE_HEX       ; wyświetl sumę

        LJMP  $
```

Przykład ten stanowi najprostszy kalkulator. Potrafi on dodać dwie liczby. Liczby te wczytywane są z klawiatury jako numer klawisza. Wykorzystywany jest do tego podprogram WAIT KAY. Oczekuje on na użycie dowolnego klawisza z klawiatury matrycowej. Po

naciśnięciu klawisza podprogram kończy swoje działanie, umieszczając w akumulatorze numer klawisza.
Klawisze [0]...[9] ponumerowane są odpowiednio 0...9, a pozostałe mają wartości 10...15, czyli 0AH...0FH. Kolejność klawiszy jest następująca:
0,1,2,3,4,5,6,7,8,9,←, →,↑ ,↓,Esc,Enter.
Kolejność ta jest zgodna z połączeniem klawiszy w systemie, co można sprawdzić na schemacie blokowym
Program ten wykonuje tylko jedno dodawanie. Aby uruchomić go ponownie i wykonać kolejne dodawanie, należy nacisnąć klawisz [RESET_RAM].
Dopóki używane są małe cyfry, wyniki na tym „ kalkulatorze" są zgodne z normalnym kalkulatorem, tj. wypisane są faktycznie w kodzie dziesiętnym. Jeżeli jednak suma cyfr będzie większa od 9 , to na wyświetlaczu oprócz cyfr pojawią się również litery A...F – czyli zapis szesnastkowy. Aby tego uniknąć, należy przed wpisaniem na wyświetlacz zamienić liczbę zapisaną w bajcie binarnie na liczbę w kodzie BCD.
Zapis w kodzie BCD polega na tym, że w jednej jednostce pamięci (np. w bajcie) pamiętana jest tylko jedna cyfra z liczby dziesiętne. Na przykład liczba 354 jest pamiętana w 3 jednostkach pamięci

- W pierwszej pamiętana jest cyfra 3
- W drugie pamiętana jest cyfra 5
- A w trzeciej pamiętana jest cyfra 4

Do zapamiętania jednej cyfry (0...9) wystarczają 4 bity pamięci. Na 4 bitach można zapamiętać 16 różnych stanów (tak jak ma to miejsce w zapisie szesnastkowym). Tutaj wykorzystane jest tylko 10stanów (0...9), natomiast pozostałe 6 jest niewykorzystanych.
Przeznaczenie całego bajtu, czyli 8 bitów, na zapamiętanie jednej cyfry jest niepotrzebną stratą miejsca. Dlatego też najczęściej stosowany zapis to tzw. „upakowane BCD". W zapisie tym na każdą cyfrę przeznaczone są 4 bity, czyli w każdym bajcie pamiętne są 2 cyfry liczby dziesiętnej. W dalszym opisie sformułowanie format BCD będzie używane w odniesieniu do formatu „upakowane BCD".
Na początku lekcji stwierdzono, że 1 bajt jest najczęściej traktowany jako liczba z zakresu 0...255. Ma to oczywiście miejsce przy formacie binarnym. Przy formacie BCD wyróżnia się tylko liczby 0...99. Zakłada się, że pozostałe stany nigdy w tych bajtach nie wystąpią. Przypadkowe ich wystąpienie(wynikające z błędu programu) spowoduje dalsze błędy przy ich interpretacji.
Należy zauważyć, że interpretacja zawartości danego bajtu, zależy od założonego formatu danych w trakcie pisania programu. Na przykład bajt zapisany w taki sposób:

0001 0000

może być odczytany jako:

- Liczba 16 przy formacie binarnym,
- Liczba 10 przy formacie BCD.

Możliwe są oczywiście jeszcze inne interpretacje. Bajty pobierane z pamięci programu są interpretowane jako odpowiednie rozkazy dla mikrokontrolera bądź jako dane tych rozkazów.
Na przykład rozkaz:

MOV A, #74H

Jest zapisany w pamięci programu jako dwa kolejne bajt, oba o wartości 74H. Z tych dwóch bajtów pierwszy jest interpretowany jako rozkaz:

MOV A, #data

A drugi stanowi dane (# data).

W przykładzie prostego kalkulatora należało wypisać na wyświetlaczu znak „+”. W tym celu użyto rozkazu:

MOV A,#'+'

który do akumulatora wprowadził kod znaku “+”, czyli jak widać w listingu, wartość 2BH. Gdyby użyć podprogramu WRITE_HEX, to na wyświetlaczu byłaby wypisana właśnie ta wartość. W celu otrzymania na wyświetlaczu znaku „+” użyto podprogramu WRITE_DATA, który zinterpretował wartość akumulatora (2BH) jako znak do wyświetlania na wyświetlaczu. Dzięki temu został wypisany znak „+”.
Jak widać z powyższych przykładów, mimo, że bajt zawiera zawsze 8 bitów, czyli 256 różnych stanów, to ich znaczenie może być bardzo różne. Wszystko zależy od zamysłu programisty.
Jeżeli wyniki obliczeń mają być przedstawione w kodzie BCD, są dwie możliwości postępowania:

- Cały czas w trakcie trwania obliczeń posługiwać się liczbami BCD
- W trakcie obliczeń posługiwać się liczbami binarnymi, na koniec zamienić liczby binarne na BCD.

Sposób pierwszy jest możliwy do zastosowania w zasadzie tylko w prostych wyliczeniach. Należy pamiętać, że ALU (jednostka arytmetyczno-logiczna) wykonuje obliczenia zawsze w ten sam sposób (prawidłowy dl liczb binarnych). Dlatego też w wyniku dodawania dwóch liczb, na przykład 05H+05H, powstanie liczba 0AH (a w formacie BCD powinno być 10H). Istnieje jednak rozkaz tzw. Poprawki dziesiętnej:

DA A

Którego użycie po powyższym dodawaniu zmieni wartość w akumulatorze z 0AH na 10H.
Rozkaz

DA A

Działa poprawnie tylko przy dodawaniu. Nie ma natomiast podobnych rozkazów dla odejmowania, mnożenia czy dzielenia. Aby poznać działanie rozkazu DA A należy dopisać go do przykładu „kalkulatora” tuż po rozkazie dodawania ADD. Po uruchomieniu tak zmienionego programu, wyniki dodawania będą przedstawione na wyświetlaczu w postaci dziesiętnej.

Jeżeli jednak zostaną użyte klawisze o numerach większych od 9, wynik dodawania będzie nieprawidłowy. Wynika to z faktu, że rozkaz DA A nie zamienia liczby binarnej na BCD. Rozkaz ten zamienia wynik dodawania dwóch liczb BCD z powrotem na liczbę BCD. Natomiast użycie klawiszy o numerach większych od 9 powoduje, iż dodawane są liczby zapisane binarnie (na przykład 0BH). Stąd wynik dodawania nie może zostać poprawiony przez rozkaz DA A.
Aby „kalkulator” działał prawidłowo dla wszystkich klawiszy, należy zamienić numery klawiszy na format BCD lub wynik dodawania, prowadzonego w formacie binarnym, zamienić na BCD.
W celu zamiany liczby binarnej na BCD najprościej użyć do tego celu dzielenia.
W mikrokontrolerze 8051 dzielenie wykonywane jest zawsze na tych samych rejestrach. Przed wykonaniem dzielenia dzielna powinna być umieszczona w akumulatorze, a dzielnik w rejestrze B (rejestr o adresie F0H w obszarze rejestrów specjalnych). Po wykonaniu rozkazu dzielenia

DIV AB

w rejestrze A otrzymamy wynik z dzielenia, natomiast w rejestrze B – resztę.

Zakładając, że liczba binarna jest mniejsza od 100, można ją zamienić na liczbę BCD poprzez podzielenie jej przez 10. Wynikiem tej operacji będzie liczba dziesiątek, natomiast resztą będzie liczba jednostek.
Przykład 3 ilustruje zamianę liczby binarnej na BCD

```
        LJMP  START
        ORG   100H
START:

        LCALL        LCD_CLR

        MOV  A, #63                  ; wpisz dzielną do A
        MOV  B, #10                  ; wpisz dzielnik do B
        DIV  AB                      ; dzielenie A / B
                                     ; wynik dzielenia w A
                                     ; - cyfra dziesiątek z 63
                                     ; reszta z dzielenia w B
                                     ; czyli cyfra jednostek
        SWAP A                       ; zamień półbajty
                                     ; -liczba dziesiątek do
                                     ; górnej połówki A
        ADD  A,B                     ; dodaj liczbę jednostek
                                     ; - liczba jednostek do
                                     ; dolnej połówki A

        LCALL        WRITE_HEX       ; wypisz liczbę BCD

        LJMP  $
```

W wyniku dzielenia liczby z akumulatora przez 10 otrzymano rozdzielenie liczby dziesiątek (A) i jednostek (B). Należy jeszcze te dwie liczby „spakować” razem do jednego bajtu. Rozkaz SWAP zamienia młodsze 4 bity akumulatora ze starszymi bitami. W ten sposób liczba dziesiątek umieszczona zostaje na właściwym miejscu, tzn. na bardziej znaczących 4 bitach. Dodanie do akumulatora reszty zapamiętanej w rejestrze B spowoduje powstanie z powrotem liczby 63 w formacie „upakowane BCD” (należy pamiętać, że przed dodawaniem 4 młodsze bity w akumulatorze i 4 starsze bity w rejestrze B są zerami, a więc w wyniku dodawania odpowiednie bity niosące informacje nie ulegną modyfikacji).
Przebieg zamiany liczby binarnej na BCD ilustruje poniższa tabela, w której przedstawiono zawartość akumulatora i rejestru B w trakcie wykonywania programu z przykładu trzeciego.

		A	**B**
		XXXX XXXX	XXXX XXXX
MOV	A,#63	0011 1111 = 63	XXXX XXXX
MOV	B,#10	0011 1111 = 63	0000 1010 = 10
DIV	A,B	0000 0110 = 6	0000 0011 = 3
SWAP	A	0110 0000 = 60_{BCD}	0000 0011 = 3
ADD	A,B	0110 0011 = 63_{BCD}	0000 0011 = 3

Po dołączeniu programu z powyższego przykładu do kalkulatora, dodawanie wartości dwóch dowolnych klawiszy będzie wykonane prawidłowo i wyświetlone zostanie w kodzie BCD.
W dotychczasowych przykładach posługiwano się liczbami mniejszymi od 256, czyli mieszczącymi się w 1 bajcie. Interesujące jest, co się stanie, jeżeli suma dwóch bajtów będzie większa od 256.
Przedstawia to poniższy przykład.

```
        LJMP  START
        ORG   100H
START:

        LCALL        LCD_CLR

        MOV   A, #250                ; wpisz do A liczbę 250
        ADD   A, #10                 ; dodaj do A liczbę 10
                                     ; w A 260 – 25 = 4
        LCALL WRITE_HEX

        LJMP  $
```

Na wyświetlaczu wypisana jest wartość 04, podczas gdy wynik dodawania wynosi 260. Należy zauważyć, że:

260=4+256=4+przeniesienie do następnego bajtu.

Jest to ta sama zasada, która obowiązuje przy dodawaniu pisemnym:

```
  19
 +15
  34
```

W tym przykładzie 9+5 nie równa się 14; 9+5 równa się 4+przeniesienie. Przeniesienie to jest uwzględniane przy dodawaniu cyfr w następnej kolumnie. Przy dodawaniu liczb w systemie dziesiętnym przeniesienie ma wartość dziesięciu jednostek danej kolumny. Natomiast przy dodawaniu całych bajtów, przeniesienie ma wartość 256 jednostek danego bajtu. Powstaje oczywiście pytanie, gdzie w mikrokontrolerze można znaleźć to przeniesienie? Jest ono umieszczone w rejestrze stanu (PSW), w bicie przeniesienia C. Jak się o tym przekonać? Można uruchomić ten przykład w trybie pracy krokowej. Po wykonaniu rozkazu ADD należy sprawdzić siódmy bit w rejestrze stanu (bit przeniesienia). Jeżeli suma dwóch liczb jest większa od 255, to bit przeniesienia C jest ustawiony na 1, w przeciwnym przypadku jest wyzerowany.
W programie bit ten może być wykorzystany na dwa sposoby. Jeżeli założono, że liczby, na których dokonywane są obliczenia nigdy nie przekroczą jednego bajtu, to ustawienie bitu C sygnalizuje błąd tych założeń. Jeżeli natomiast prowadzone są obliczenia na liczbach kilkubajtowych, bit ten powinien być uwzględniony, tak jak przy dodawaniu pisemnym, czyli przy dodawaniu kolejnych bajtów. Służy do tego specjalny rozkaz:

ADDC A, ...

który do dwóch składników dodaje jeszcze bit przeniesienia (0 lub 1). Poza tą różnicą rozkaz ADDC działa dokładnie tak samo, jak rozkaz ADD. Rozkaz ADDC umożliwia dodawanie liczb wielobajtowych.
Równie podstawowym działaniem jak dodawanie, jest odejmowanie. W mikrokontrolerze 8051 jest do dyspozycji tylko jeden rozkaz:

SUBB A, ...

który od zawartości akumulatora odejmuje drugi argument oraz zawartość flagi C. Flaga C przy odejmowaniu pełni rolę pożyczki. Ponieważ nie ma rozkazu odejmowania nie uwzględniającego flagi C, zawsze przed rozpoczęciem odejmowania należy wyzerować flagę C.

```
        LJMP  START
        ORG   100H
START:
        LCALL       LCD_CLR

        MOV   A, #6                 ; wpisz do A liczbę 6
        CLR   C                     ; zeruj bit przeniesienia
        SUBB  A, #1                 ; odejmij z pożyczką
                                    ; A <- A-1-C = 6-1-0 = 5
        LCALL       WRITE_HEX

        MOV   A, #6                 ; wpisz do A liczbę 6
        SETB  C                     ; ustaw bit przeniesienia
        SUBB  A, #1                 ; odejmij z pożyczką
                                    ; A <- A-1-C = 6-1-1 = 4
        LCALL       WRITE_HEX

        LJMP  $
```

Jak widać, po uruchomieniu przykładu wynik odejmowania: 6-1 jest różny, w zależności od ustawienia flagi C. Wynik prawidłowy jest oczywiście, gdy flaga C jest równa 0.
Wraz z odejmowaniem pojawia się problem liczb ujemnych. Jeżeli zamiast 6-1 zostanie wykonane działanie 1-6, to w wyniku powinna powstać liczba: -5. Tylko jak powinna ona zostać zapisana? Do tej pory liczba zapisana w jednym bajcie była interpretowania jako liczba z zakresu 0...255. Nie ma tu miejsca na znak „-„.
Rozwiązań tego problemu można znaleźć wiele. Można na przykład przeznaczyć dodatkowy bajt na zapamiętanie znaku liczby lub ograniczyć liczby do 7 bitów (0...127), a na 8 bicie zapisywać znak, przykładowo 0 to „+”, 1 to „-„.
Każde z tych rozwiązań jest dobre, pod warunkiem, że operacje arytmetyczne będą prawidłowo wykonywane na tych liczbach. Ideałem byłoby, gdyby zarówno do liczb dodatnich, jak i ujemnych, można było zastosować te same rozkazy w celu ich dodawania lub odejmowania. W oparciu o to założenie powstał format danych zwany uzupełnieniem do 2 (oznaczany symbolem U2). W formacie tym liczby zapisane na jednym bajcie mogą być z zakresu –128...+127. Jak wspomniano, dodawanie i odejmowanie tych liczb powinno być prawidłowo wykonane za pomocą normalnych rozkazów arytmetycznych.
Ponieważ musi być spełniona zależność: -1+1=0, więc –1=0-1.

```
        LJMP  START
        ORG   100H
START:
        LCALL       LCD_CLR

        CLR   A                     ; zeruj A
        CLR   C                     ; zeruj C
        SUBB  A, #1                 ; A ← 0 – 1 = - 1
        MOV   R0, A                 ; zapamiętaj w R0
        LCALL       WRITE_HEX

        MOV   A, R0                 ; A ← R0 = - 1
        ADD   A, #1                 ; A ← A + 1 = -1 + 1 = 0
        LCALL       WRITE_HEX

        SJMP  $
```

Jak widać, po uruchomieniu tego przykładu liczba –1 jest zapisana jako 0FFH. Dodanie do niej, za pomocą poznanego już rozkazu ADD, liczby 1 daje w wyniku wartość 0, zgodnie z założeniem.
Można łatwo się przekonać, że kolejne liczby ujemne będą reprezentowane na 1 bajcie następująco:

-1 0FFH
-2 0FFH
-3 0FDH.

Można też sprawdzić, że prawidłowo będą wykonane proste operacje, na przykład:

-1 + (-1) = -2 [0FFH + 0FFH = 0FEH].

Na końcu tego przykładu w celu zbudowania pustej pętli użyto rozkaz SJMP, zamiast dotychczas stosowanego rozkazu LJMP. Jak widać, rozkaz ten zajmuje tylko 2 bajty, podczas gdy LJMP zajmował 3. W rozkazie SJMP adres, do którego należy wykonać skok, jest określony względem obecnego położenia i na określenie tego przesunięcia użyto tylko 1 bajtu.
Rozkaz SJMP $, to skok do tego samego adresu, pod którym w pamięci programu rozkaz ten się zaczyna. A więc przesunięcie powinno równać się 0. Najpierw pobierane są bajty rozkazu i licznik rozkazów jest zwiększany tak, aby wskazywał kolejny rozkaz. Dopiero potem następuje wykonanie rozkazu. W tej sytuacji dla rozkazu 2-bajtowego, rozkaz skoku na adres tego rozkazu oznacza skok o –2. Bajt 0FEH umieszczony w rozkazie SJMP jest interpretowany jako liczba w kodzie U2. Dzięki temu, wszystkie rozkazy skoków używające adresowania względnego mogą być wykonane w zakresie –128...127 względem pierwszego bajtu kolejnego rozkazu.
W zapisie uzupełnienia do 2 wszystkie ujemne liczby mają najstarszy bit ustawiony na 1, natomiast liczb dodatnich najstarszy bit ustawiony jest na 0. Przy posługiwaniu się liczbami w kodzie uzupełnienia do 2, czyli z zakresu –128...127, powstaje problem dodania dwóch liczb mniejszych od 127, których suma jest większa od 127. Przy dodawaniu, na przykład 100+100=200, uzyskany wynik jest większy od 127, czyli ósmy bit jest ustawiony na 1. Interpretując tę liczbę jako zapisaną w kodzie uzupełnienie do 2, odczytać ją należy jako liczbę ujemną, a dokładnie jako –56. Aby zdarzenie takie mogło być łatwo zauważone, istnieje w mikrokontrolerze specjalna flaga: OV (overflow – przepełnienie), która sygnalizuje wystąpienie tego typu procesów. Flaga ta umieszczona w rejestrze stanu powinna być

kontrolowana po każdej operacji dodawania lub odejmowania, przy posługiwaniu się liczbami zapisywanymi w kodzie uzupełnienia do 2.
Oprócz omówionych już działań arytmetycznych (dodawanie, odejmowanie, dzielenie), mikrokontroler potrafi również realizować mnożenie. Służy do tego rozkaz MUL AB. Tak jak w dzieleni, mnożenie wykonywane jest zawsze na dwóch tych samych rejestrach: akumulatorze i rejestrze B. Wynik mnożenia dwóch liczb 8-bitowych może być 16-bitowy. Mniej znacząca część wyniku umieszczona jest w akumulatorze, natomiast bardziej znacząca w rejestrze B.

```
        LJMP  START
        ORG   100H
START:

        LCALL       LCD_CLR

        MOV  A, #0F1H                ; mnożna
        MOV  B, #2                   ; mnożnik
        MUL  AB                      ; mnożenia A*B
                                     ; starsza część wyniku w B
                                     ; młodsza część wyniku w A
        XCH  A, B
        LCALL       WRITE_HEX        ; wypisz starszą część

        MOV  A, B                    ; pobierz młodszą część
        LCALL       WRITE_HEX        ; wypisz młodszą część

        SJMP  $
```

Na wyświetlaczu przedstawiony jest najpierw starszy bajt wyniku, a potem młodszy. Przy okazji można zauważyć, że mnożenie liczby binarnej przez 2 sprowadza się do przesunięcia wszystkich bitów o jeden w lewo i dopisania 0 na najmniej znaczący bit.
W przykładzie wykorzystano nową instrukcję: XCH (exchange - wymiana). Zamienia ona zawartość dwóch argumentów między sobą, przy czym jednym z nich jest zawsze akumulator. Użycie tej instrukcji w przykładzie pozwala na wyświetlenie w pierwszej kolejności starszej części wyniku, bez wykorzystywania dodatkowych rejestrów.
Należy zaznaczyć, że rozkazy mnożenia i dzielenia są prawidłowo wykonywane tylko dla liczb binarnych. Użycie tych rozkazów dla liczb w kodzie uzupełnienia do 2 lub w BCD da błędne wyniki.
Podsumowując, należy zauważyć, że mikrokontroler pozwala na bezpośrednią realizujących działań na liczbach jednobajtowych:

- przy liczbach 0...255:
  - dodawanie ADD,ADDC,
  - odejmowanie SUBB,
  - mnożenie MUL,
  - dzielenie DIV,
- przy liczbach –128..127 w kodzie U2:
  - dodawanie ADD, ADDC,
  - odejmowanie SUBB,
- przy liczbach 0...99 w kodzie BCD:

- dodawanie ADD, ADDC + rozkaz korekcji dziesiętnej DAA.

Planując sposoby obliczeń i reprezentacji liczb w programie należy o tym pamiętać.

ZADANIA

ZADANIE 1

Korzystając z przykładów w lekcji napisać program „kalkulatora" dodającego numery klawiszy i przedstawiającego wszystkie liczby w kodach BCD.

ZADANIE 2

Napisać program „kalkulatora" mnożącego liczby w postaci BCD.

WSKAZÓWKI

Ad. 1

Aby przedstawić wszystkie liczby w kodach BCD, należy po pobraniu numeru klawisza zamienić go na BCD. Dzięki temu, dodawane są dwie liczby BCD. Wystarczy zatem po ich dodaniu zastosować rozkaz poprawki dziesiętnej akumulatora.

```
        LJMP  START
        ORG   100H
START:

        LCALL         LCD_CLR

        LCALL         WAIT_KEY          ; pobierz pierwszy czynnik
        MOV   B, #10                    ; zamień liczbę na BCD
        DIV   AB                        ; dzieląc przez 10
        SWAP A
        ADD   A,B
        MOV   R0, A                     ; zapamiętaj w R0 (BCD)
        LCALL         WRITE_HEX         ; wypisz na LCD

        MOV   A, #'+'                   : znak sumy
        LCALL         WRITE_DATA

        LCALL         WAIT_KEY          ; pobierz drugi czynnik
        MOV   B, #10                    ; zamień liczbę na BCD
        DIV   AB                        ; dzieląc przez 10
        SWAP A
        ADD   A,B
        MOV   R0, A                     ; zapamiętaj w R1 (BCD)
        LCALL         WRITE_HEX         ; wypisz na LCD

        MOV   A, #'='                   ; znak równości
```

```
LCALL     WRITE_DATA     ; wyświetl jako znak

MOV  A, R0               ; pierwszy czynnik do A
ADD  A, R1               ; dodaj drugi czynnik
DA   A                   ; poprawka dodawania
                         ; liczb BCD

LCALL     WRITE_HEX      ; wypisz czynnik na LCD

SJMP  $
```

Ad. 2

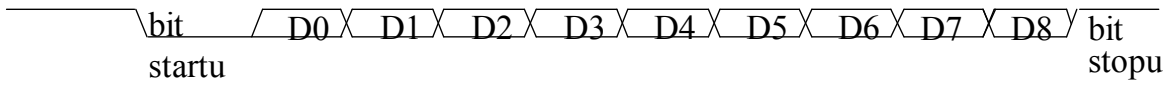
Można skorzystać z przykładu do poprzedniego zadania. Oprócz oczywistej zamiany dodawania na mnożenie, należy jeszcze zmienić sposób przechowywania czynników do mnożenia. Rozkaz MUL wykonywany jest prawidłowo jedynie dla liczb binarnych. Dlatego też numery klawiszy należy zapamiętać w odpowiednich rejestrach, przed ich zamianą na kod BCD.
Po końcowym pomnożeniu tych liczb, trzeba jeszcze wynik mnożenia, przed wyświetleniem na LCD, zamienić na kod BCD. Ponieważ sposób zamiany, przedstawiony wyżej, zamienia prawidłowo jedynie liczby z zakresu 0...99, to jeśli wynik mnożenia będzie większy od 100, kalkulator nie pokaże prawidłowego wyniku.
Taki program, nie najlepiej działający, zamieszczony jest na dyskietce w postaci przykładu 9 do tej lekcji.
Prawidłowe rozwiązanie tego zadania jest przedstawione w lekcji 5.

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ĆWICZENIE 12

Transmisja szeregowa.

W systemach mikroprocesorowych bardzo często zachodzi konieczność przesłania danych od innych systemów lub do komputerów. Najczęściej potrzebna jest transmisja w obu kierunkach. Typowym przykładem może być tu system pomiarowy, który wykonuje pomiary na rozkaz przesłany z komputera, a następnie odsyła wyniki pomiarów.
W mikroprocesorach 8-bitowych z pozoru najprościej jest przesyłać dane całymi bajtami jednocześnie. Jest to tzw. transmisja równoległa. Każdemu bitowi przyporządkowana jest jedna linia, tak więc na cały bajt potrzeba 8 linii mikrokontrolera. Dodatkowo potrzebne są jeszcze linie sterujące. Liczba linii wejść/wyjść mikrokontrolera, które trzeba przeznaczyć na ten rodzaj transmisji, jest dość znaczna. W systemach mikroprocesorowych jest to duża niedogodność. Przeważnie nie ma tylu wolnych linii w mikrokontrolerze, więc trzeba dobudować układy zewnętrzne. W systemie DSM-51 możliwa jest transmisja równoległa poprzez zewnętrzny układ 8255.
Drugą poważną niedogodnością tego typu transmisji jest liczba przewodów potrzebnych do połączenia. Przy transmisji na większe odległości powstają problemy przesłuchów pomiędzy liniami, co stwarza konieczność dodatkowego ekranowania i obniżenia prędkości transmisji.
Z tych powodów transmisja równoległa jest praktycznie wykorzystywana tylko do przesyłania danych na niedużą odległość (z komputera do drukarki). Przy łączeniu systemów mikroprocesorowych zdecydowanie króluje transmisja szeregowa.
W transmisji szeregowej bity przesyłane są szeregowo jeden za drugim. Istnieją dwa sposoby transmisji szeregowej: synchroniczna i asynchroniczna.
W transmisji synchronicznej, oprócz linii danych, po której przesyłane są kolejne bity danych, istnieje jeszcze linii synchronizacji, po której przesyłane są impulsy informujące, w których momentach na linii danych jest kolejny bit. Do tego sposobu transmisji potrzebne są dwie linie.
Zdecydowanie najczęściej wykorzystywana jest transmisja asynchroniczna. Potrzebna jest do niej jedna linia. Dla rozróżnienia kolejnych bajtów przesyła się dodatkowo specjalne bity sterujące. Przesłanie jednego bajtu wygląda następująco:



Pomiędzy transmisją kolejnych bajtów linia jest w stanie wysokim. Transmisja bajtu rozpoczyna się od wysłania bitu startu, który zawsze jest równy 0. Następnie przesyłane są kolejne bity bajtu, w kolejności od najmłodszego do najstarszego. Po danych wysyłany jest bit parzystości. Jego wartość zależy od liczby bitów równych 1 w przesyłanym bajcie i służy do kontroli poprawności transmisji. Bit parzystości może kontrolować parzystość, nieparzystość lub być w ogóle pominięty. Na koniec przesyłane są (1 lub 2) bity stopu. Bity te mają wartość 1, a więc ustawiają już linię w stan stabilny, który występuje pomiędzy transmisją poszczególnych bajtów.
Sposób przesyłania jednego bajtu musi być jednakowo zdefiniowany w nadajniku i odbiorniku przed rozpoczęciem transmisji. W przeciwnym razie transmisja może być niezrozumiała. Oprócz ustalenia przesyłanych bitów trzeba jeszcze zdefiniować prędkość transmisji.

Prędkość transmisji wyrażona jest w bodach, czyli w liczbie bitów transmitowanych w ciągu 1 sekundy. Istnieją typowe ustalone prędkości transmisji. Są to, zaczynając od 300 bodów, prędkość uzyskane przez kolejne podwajanie tej liczby, a więc: 300, 600, 1200, 2400, 4800, 9600, 19200.
Prędkość 19200 bodów jest w zasadzie maksymalną prędkością w standardzie RS232. Jednak przy niedużych odległościach można stosować wyższe prędkości. Sterownik transmisji RS232 umieszczony w komputerze IBM PC może prowadzić transmisję z maksymalną prędkością 115200 bodów. W systemie DSM-51 maksymalna prędkość wynosi 57600. Taka też prędkość jest wykorzystywana do przesyłania programów z komputera do systemu DSM-51.
Należy zaznaczyć, że prędkość przesyłania bajtów nie wynika z podzielenia przez 8 prędkości wyrażonej w bodach. Do przesłania 1 bajtu zużywa się minimum 10 bitów (8bitów+bit startu +bit stopu)., a maksymalnie 12 bitów (dodatkowo bit parzystości i drugi bit stopu).
W mikrokontrolerze 8051 wbudowano do wnętrza sterownik transmisji szeregowej. Może on pracować w czterech trybach, z czego tryb 0 to transmisja synchroniczna, a tryby 1...3 to transmisja asynchroniczna. W systemie DSM-51 można wykorzystać transmisję asynchroniczną, a więc tryb 1...3.
Sterowanie transmisją szeregową odbywa się poprzez wpisanie odpowiedniego bajtu do rejestru SCON, który znajduje się w obszarze rejestrów specjalnych, pod adresem 98H. Rejestr ten wygląda następująco:

SM0	SM1	SM3	REN	TB8	RB8	TI	RI

SM0, SM1 Ustawienie trybu transmisji:

SM0	SM1	Tryb	Transmisja	Prędkość
0	0	0	synchroniczna	$f_{OSC}/12$
0	1	1	asynchroniczna 8 bit	Timer 1
1	0	2	asynchroniczna 9 bit	$f_{OSC}/64$ lub $f_{OSC}/32$
1	1	3	asynchroniczna 9 bit	Timer 1

SM2 sterowanie komunikacji wieloprocesorowej w trybach 2 i 3 (normalnie = 0)
REN zezwolenie na odbiór. Jeśli wpisane jest 0, sterownik tylko nadaje,
TB8, RB8 9 bit transmisji w trybie 2 i 3 odpowiednio dla nadawania i odbioru,
T1, R1 flagi zakończenia operacji nadawania / odbioru.

Standardowo do transmisji komputer – DSM-51 wykorzystuje się tryb 1. W trybie tym przesyłany jest bit startu, 8 bitów danych i bit stopu.
Do rejestru SCON należy wpisać wartość 0101 0000B. Jak widać z tabeli, prędkość transmisji ustalana jest przez Timer 1, a dokładnie jest określona wzorem:

$V = [\ 2^{SMOD} / 32\ ] * [\ 1 / \text{okres Timera 1}\ ]$.

SMOD jest najstarszym bitem w rejestrze PCON. Ponieważ bity tego rejestru nie mogą być indywidualnie adresowane, do ustawienia bitu SMOD trzeba użyć odpowiednio rozkazów ANL i ORL. Timer 1 używany jest najczęściej w trybie 2 – pracuje wtedy jako automatycznie przeładowywany timer 8-bitowy. Przy tym założeniu wzór na prędkość wygląda tak:

$V = [\ 2^{SMOD} / 32\ ] * [\ f_{OSC} / (\ 12 * [\ 256 - TH1\ ]\ )\ ].$

gdzie TH1 – wartość wpisana do rejestru TH1.
Stąd:

$$256 - TH1 = [\ 2^{SMOD} / 32\ ] * [\ 11.059.200 / (\ 12 * V\ )\ ] =$$
$$= [\ 2^{SMOD} * 11.059.200\ ] / [\ 32 * 12 * V\ ] =$$
$$= [\ 2^{SMOD} * 28800\ ] / V.$$

Dla SMOD = 1 otrzymujemy

$$TH1 = 256 - [\ 57600 / V\ ].$$

Tutaj ujawnia się druga zaleta zastosowanego w systemie DSM-51 rezonatora kwarcowego. Wszystkie wymienione wyżej standardowe prędkości transmisji mogą być w sterowniku mikrokontrolera 8051 dokładnie ustawione. Dodatkowo równie dokładnie można ustawić prędkość 19200 * 3 = 57600, dostępną również w komputerach IBM PC. Trzeba jasno powiedzieć, że to właśnie możliwość ustawiania standardowych prędkości transmisji szeregowej zadecydowała o wyborze takiego rezonatora.

Prostym przykładem transmisji szeregowej jest poniższy program.

```
;************* Ustawienie TIMERów ************

;TIMER 0
T0_G  EQU   0                    ; GATE
T0_C  EQU   0                    ; COUNTER/-TIMER
T0_M  EQU   1                    ; MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8
; TIMER 1
T1_G  EQU   0                    ;GATE
T1_C  EQU   0                    ; COUNTER/-TIMER
T1_M  EQU   0                    ; MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

;**************Transmisja szeregowa***************
TR_M EQU   1                     ; tryb transmisji (1...3)
TR_R  EQU   0                    ; zezwolenie na odbiór

SCON_SET   EQU   TR_M*64+TR_R*16

; SMOD=1
; TIMER1=57600/300bodów=192
TH1_SET      EQU   256-192
TL1_SET      EQU   256-192
;*********************************************
```

```
        LJMP  START
        ORG   100H
START:
        MOV   SCON, #SCON_SET          ; port szeregowy
        ORL   PCON, #80H               ; SMOD=1
        MOV   TMOD, #TMOD_SET          ; Timer 1 dla
        MOV   TH1, #TH1_SET            ; transmisji
        MOV   TL1, #TL1_SET
        SETB  TR1                      ; start Timera 1

        SETB  TI
        LCALL       LCD_CLR
LOOP:
        LCALL       WAIT_KEY
        JNB   TI, $                    ; czy można nadać
        CLR   TI                       ; kolejny znak
        ADD   A, #30H
        MOV   SBUF, A                  ; nadaj znak
        LCALL       WRITE_DATA         ; wyświetl na LCD
        SJMP  LOOP
```

Po ustawieniach rejestru SCON i Timera 1, system DSM-51 nadaje kolejno kody naciskanych klawiszy. Prędkość transmisji została ustawiona na 300 bodów. Aby to osiągnąć, należy zgodnie ze wzorem ustawić okres Timera 1 = 192 (dla SMOD = 1). Pamiętając, że timer liczy w górę do wartości 256, należy do rejestru TH1 wpisać wartości 256-192. Nie trzeba włączać przerwań od Timera 1. Dla sterownika transmisji wystarczający jest sam sygnał przepełnienia Timera 1.
Zapoczątkowanie transmisji następuje w momencie wpisania bajtu do rejestru SBUF. Sterownik automatycznie wysyła bajt z bufora transmisji (SBUF) szeregowo przez linię TxD. Sterownik sygnalizuje zakończenie transmisji bajtu poprzez ustawienie flagi T1. Od tej pory można wpisać kolejny bajt do bufora transmisji. Wpisanie kolejnego bajtu przed zakończeniem transmisji poprzedniego spowodowałoby zapisanie nowego bajtu na częściowo wysunięty poprzedni – wystąpiłby błąd w transmisji. Dlatego też każdorazowo przed wpisaniem bajtu do SBUF należy sprawdzić stan flagi T1.
Działanie przykładu można zaobserwować włączając na komputerze IBM PC dowolny program emulujący terminal. Jeżeli ustawienia będą zgodne z przyjętymi w programie, to na ekranie komputera pojawią się znaki odpowiadające wybranym klawiszom.
Przykład 2 ilustruje odbiór transmisji szeregowej.

```
;************** Ustawienie TIMERów ************
;TIMER 0
T0_G  EQU   0                  ; GATE
T0_C  EQU   0                  ; COUNTER/-TIMER
T0_M  EQU   1                  ; MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8
; TIMER 1
T1_G  EQU   0                  ;GATE
T1_C  EQU   0                  ; COUNTER/-TIMER
T1_M  EQU   0                  ; MODE (0..3)
```

```
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

;***********Transmisja Szeregowa***************
TR_M EQU   1                    ; tryb transmisji (1...3)
TR_R  EQU   0                   ; zezwolenie na odbiór

SCON_SET   EQU   TR_M*64+TR_R*16

; SMOD=1
; TIMER1=57600/300bodów=192
TH1_SET      EQU   256-192
TL1_SET      EQU   256-192
;*******************************************

        LJMP  START
        ORG   100H
START:
        MOV   SCON, #SCON_SET           ; port szeregowy
        ORL   PCON, #80H                ; SMOD=1
        MOV   TMOD, #TMOD_SET           ; Timer 1 dla
        MOV   TH1, #TH1_SET             ; transmisji
        MOV   TL1, #TL1_SET
        SETB  TR1                       ; start Timera 1

        LCALL        LCD_CLR
LOOP:
        JNB   RI, $                     ; czy odebrany znak
        CLR   RI
        MOV   A, SBUF                   ; pobierz znak
        LCALL        WRITE_DATA         ; wyświetl na LCD
        SJMP  LOOP
```

W momencie wystąpienia bitu startu sterownik automatycznie rozpoczyna odbiór transmisji. Po skompletowaniu całego bajtu (zgodnie z ustawioną prędkością transmisji) sterownik przepisuje bajt do bufora transmisji SBUF. Jednocześnie sygnalizuje ten stan poprzez ustawienie flagi RI.
Odbiór transmisji w programie polega na odczytaniu rejestru SBUF po ustawieniu flagi RI. Następnie flagę należy wyzerować, aby sterownik mógł sygnalizować odebranie kolejnego bajtu. Zawartość rejestru SBUF jest prawidłowa, aż do momentu zakończenia odbioru kolejnego bajtu przez sterownik. W tym momencie nowy bajt jest wpisywany na miejsce starego. Tak więc program powinien zdążyć odczytać bajt przed odebraniem następnego.
Program można uruchomić łącząc DSM-51 z komputerem. Druga możliwość to podłączenie dwóch systemów DSM-51 poprzez COM1 za pomocą kabla RS232. Na jednym należy uruchomić program z przykładu 1, a na drugim z przykładu 2.
Rejestr SBUF z przykładu 2 nie jest, pomimo jednej nazwy, tym samym rejestrem, co SBUF z przykładu 1. Chociaż adres tego rejestru jest zawsze taki sam, w rzeczywistości są to dwa rejestry. Do jednego z nich można tylko pisać (bufor nadawczy), natomiast drugi może być

tylko czytany (bufor odbiorczy). Tak więc bajty nadawane i odbierane nie mieszają się ze sobą.
Kolejny przykład realizuje jednoczesną transmisję w obu kierunkach. Aby ułatwić jej obsługę, zostały wykorzystane przerwania.

```
;************* Ustawienie TIMERów ************
;TIMER 0
T0_G  EQU   0                       ; GATE
T0_C  EQU   0                       ; COUNTER/-TIMER
T0_M  EQU   1                       ; MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8
; TIMER 1
T1_G  EQU   0                       ;GATE
T1_C  EQU   0                       ; COUNTER/-TIMER
T1_M  EQU   0                       ; MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

;***************Transmisja szeregowa***************
TR_M EQU   1                        ; tryb transmisji (1...3)
TR_R  EQU   0                       ; zezwolenie na odbiór

SCON_SET   EQU   TR_M*64+TR_R*16

; SMOD=1
; TIMER1=57600/300bodów=192
TH1_SET      EQU   256-192
TL1_SET      EQU   256-192
;*********************************************

        LJMP  START

;***************Przerwanie RS 232 ****************
        ORG   23H
        PUSH  ACC
        PUSH  PSW
        JBC   TI, NAD                       ; koniec nadania znaku
        CLR   RI                            ; znak odebrany
        MOV   A, SBUF                       ; pobranie znaku
        LCALL         WRITE_DATA            ; wyświetl na LCD
NAD:
        POP   PSW
        POP   ACC
        RETI

;*********************************************
        ORG   100H
START:
```

```
        MOV  SCON, #SCON_SET          ; port szeregowy
        ORL  PCON, #80H               ; SMOD=1
        MOV  TMOD, #TMOD_SET          ; Timer 1 dla
        MOV  TH1, #TH1_SET            ; transmisji
        MOV  TL1, #TL1_SET
        SETB TR1                      ; start Timera 1

        SETB EA                       ; zezwolenie na przerwanie
        SETB ES                       ; z transmisji szeregowej

        LCALL      LCD_CLR
LOOP:
        LCALL      WAIT_KEY           ; czekaj na klawisz
        ADD  A, #30H                  ; modyfikuj
        MOV  SBUF, A                  ; nadaj znak
        SJMP LOOP
```

Przerwanie od transmisji zgłaszane jest w momencie ustawiania flagi T1 lub R1, a więc zakończenia nadawania lub odbioru. Obsługę przerwania należy umieścić pod adresem 23H. Rozróżnienie, czy przerwanie związane jest z nadawaniem, czy z odbiorem opiera się na sprawdzeniu flagi TI i RI. W trakcie obsługi przerwania odpowiednia flaga powinna być wyzerowana.
W powyższym przykładzie odbiór transmisji obsługiwany jest całkowicie w przerwaniu. Zgłoszenie skompletowanego bajtu w buforze odbiorczym przez flagę RI powoduje jego odczytanie i wypisanie na wyświetlacz LCD. Obsługa przerwania pochodzącego od flagi TI, czyli zakończenie nadawania bajtu, została sprowadzona jedynie do zerowania tej flagi. Natomiast wysłanie kolejnego bajtu odbywa się z programu głównego. Założono, że zanim zostanie ponownie użyta klawiatura, poprzedni znak zostanie nadany w całości.
Taka organizacja transmisji nie jest zupełnie prawidłowa. Umieszczone wewnątrz przerwania wypisywanie znaku na wyświetlacz LCD powoduje niepotrzebne wydłużenie obsługi przerwania. Jednocześnie to nie przerwanie, a program główny powinien decydować, co zrobić z odbieranymi danymi. Niektóre mogą być w rzeczywistości wyświetlane bezpośrednio na wyświetlaczu LCD jako przesyłane komunikaty, ale inne mogą pełnić ma przykład funkcje sterujące programem głównym.
Również nadawanie powinno być oddzielone od programu głównego. Często program musi nadać nie jeden, ale cały ciąg znaków. Nadając kolejne znaki i czekając na flagę TI program główny będzie całkowicie zajęty tym zadaniem, podczas gdy faktycznie mógłby wykonywać już inne pożyteczne czynności.
Takie rozdzielenie transmisji od programu głównego realizowane jest poprzez bufory: nadawczy i odbiorczy. Bufory są w tym przypadku nie pojedynczymi rejestrami, ale kilkoma lub kilkunastoma komórkami pamięci. Odbierane bajty wpisywane są do kolejnych komórek pamięci bufora odbiorczego. Program główny odczytuje te bajty kolejno, zgodnie z zapotrzebowaniem. Bufor nadawczy wykorzystywany jest analogicznie do przekazywania danych z programu głównego do systemu nadawczego.
Obrazuje to przykład 4.

```
;**************Pamięć wewnętrzna RAM ***************
B1R2  EQU  8+2                        ; rejestr 2 w banku 1
B1R3  EQU  8+3                        ; rejestr 3 w banku 1
```

```
FLAGA       EQU   20H                ; zestaw flag bitowych
NADAJE EQU        FLAGA.0            ; trwa nadawanie znaku
BUFOR       EQU   10H                ; wielkość bufora

BUF_N       EQU   30H                ; bufor nadawczy
BUF_NE EQU        BUF_N+BUFOR        ; koniec buf. nad.
BUF_O EQU BUF_NE                     ; bufor odbiorczy
BUF_OE EQU        BUF_O+BUFOR        ; koniec buf. odb.

STOS  EQU   60H                      ; pozycja stosu

;************** Ustawienie TIMERów ************
;TIMER 0
T0_G  EQU   0                        ; GATE
T0_C  EQU   0                        ; COUNTER/-TIMER
T0_M  EQU   1                        ; MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8
; TIMER 1
T1_G  EQU   0                        ;GATE
T1_C  EQU   0                        ; COUNTER/-TIMER
T1_M  EQU   0                        ; MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET  EQU   TIM0+TIM1*16

;***************Transmisja szeregowa***************
TR_M EQU  1                          ; tryb transmisji (1...3)
TR_R  EQU   0                        ; zezwolenie na odbiór

SCON_SET   EQU   TR_M*64+TR_R*16

; SMOD=1
; TIMER1=57600/300bodów=192
TH1_SET     EQU   256-192
TL1_SET     EQU   256-192

;*************** MACRA ************************
BANK0       MACRO                    ; ustawienie banku 0 rejestrów
        CLR   RS0                    ; (z banku 0)
        MACEND

BANK1       MACRO                    ; ustawienie banku 1 rejestrów
        SETB  RS0                    ; (z banku 0)
        MACEND

;**********************************************
        LJMP  START
```

```
;*************** Przerwanie RS 232 ****************

        ORG   23H
        PUSH  ACC
        PUSH  PSW
        BANK1
        SETB  RS0
        MACEND

        JBC   TI, NAD                 ; koniec nadania znaku
        CLR   RI                      ; znak odebrany
        MOV   @R1, SBUF               ; wpisz do buf. odb.
        INC   R3                      ; zwiększ licznik
        INC   R1                      ; zwiększ adres wpisu
        CJNE  R1, #BUF_OE, ODB_E
        MOV   R1, #BUF_O              ; zapętlenie bufora

ODB_E:
        POP   PSW
        POP   ACC
        RETI

NAD:                                  ; nadaj następny znak
        CJNE  R2, #0, NAD_N           ; czy jest w buforze

        CLR   NADAJE                  ; nie – koniec nadawania
        POP   PSW
        POP   ACC
        RETI

NAD_N:
        MOV   SBUF, @R0               ; nadanie następnego znaku
        DEC   R2                      ; zmniejsz licznik
        INC   R0                      ; zwiększ adres pobierania
        CJNE  R0, #BUF_NE, NAD_E
        MOV   R0, #BUF_N              ; zapętlenie bufora
NAD_E:
        POP   PSW
        POP   ACC
        RETI

;*******************************************
        ORG   100H
START:
        MOV   SP, #STOS

        MOV   SCON, #SCON_SET         ; port szeregowy
        ORL   PCON, #80H              ; SMOD=1
        MOV   TMOD, #TMOD_SET         ; Timer 1 dla
```

```
        MOV  TH1, #TH1_SET          ; transmisji
        MOV  TL1, #TL1_SET
        SETB TR1                    ; start Timera 1

        CLR  NADAJE                 ; nie trwa nadawanie

        MOV  R0, #BUF_N             ; ustawienie adresów
        MOV  R1, #BUF_O             ; wpisu i pobierania
        BANK1                       ; dla buforów
        SETB RS0                    ; (z banku 0)
        MACEND
        MOV  R0, #BUF_N
        MOV  R1, #BUF_O
        MOV  R2, #0
        MOV  R3, #0
        BANK0
        CLR  RS0                    ; (z banku 1)
        MASCEND
        SETB EA                     ; zezwolenie na przerwanie
        SETB ES                     ; z transmisji szeregowej

        LCALL      LCD_CLR
LOOP:
        MOV  A, B1R3                ; czy bufor odbioru pusty
        JZ   ODB_NO
        MOV  A, @R1                 ; nie – wpisz znak
        LCALL      WRITE_DATA
        DEC  B1R3                   ; zmniejsz licznik
        INC  R1                     ; zwiększ adres pobierania
        CJNE R1, #BUF_OE, LOOP
        MOV  R1, #BUF_O             ; zapętlenie bufora
        SJMP LOOP

ODB_NO:
        LCALL      TEST_ENTER       ; czy trzeba nadawać
        JC   LOOP

        MOV  DPTR, #TEXT            ; nadaj text
LP1:
        CLR  A
        MOVC       A, @A+DPTR       ; pobierz znak
        JZ   LOOP                   ; 0 – koniec textu

        PUSH ACC                    ; czy jest miejsce
        MOV  A, #BUFOR              ; w buforze nadawczym
LP2:
        CJNE A, B1R2, LP3
        SJMP LP2
LP3:
```

```
        POP   ACC

        MOV   @R0, A              ; wpisz znak do bufora
        INC   R0                  ; zwiększ adres wpisu
        INC   B1R2                ; zwiększ licznik

        JB    NADAJE, LP4         ; inicjuj nadawanie
        SETB  NADAJE              ; jeśli nie trwa
        SETB  TI
LP4:
        INC   DPTR                ; następny znak w tekście
        CJNE  R0, #BUF_NE, LP1
        MOV   R0, #BUF_N          ; zapętlenie bufora nad.
        SJMP  LP1

TEXT:
        DB    'MicroMade', 0
```

Wpis do bufora odbiorczego następuje w przerwaniu, przy wykorzystaniu rejestru R1 z banku 1. Po każdym wpisuje zwiększany jest adres zawarty w rejestrze R1 oraz licznik w rejestrze R3 z banku 1. W programie głównym następuje sprawdzenie licznika. Jeśli jest on różny od zera, to znaczy, że w buforze są bajty do odebrania. Należy więc je pobrać i wyświetlić na wyświetlaczu, odpowiednio korygując licznik oraz wskaźnik odczytu z bufora – rejestr R1 z banku 0.
bufor odbiorczy (również nadawczy) nie jest nieskończony. Przeznaczony jest dla niego pewien obszar pamięci RAM. Jeżeli wskaźnik dojdzie do końca tego obszaru, to musi być z powrotem przestawiony na jego początek. Dopóki liczba bajtów w buforze nie przekroczy jego pojemności, wszystko będzie w porządku. W buforze odbiorczym nie jest to kontrolowane, gdyż i tak program nie ma wpływu na liczbę bajtów nadawanych z zewnątrz. Natomiast w buforze nadawczym przed wpisaniem kolejnego bajtu sprawdzana jest liczba batów w buforze. Jeżeli liczba ta równa się pojemności bufora, program musi zaczekać, aż zostanie nadany kolejny bajt i zwolni się miejsce w buforze.
Przy nadawaniu powstaje jeszcze problem rozpoczęcia nadawania. Zazwyczaj po zakończeniu nadawania bajtu następuje ustawienie flagi TI i w przerwaniu rozpoczyna się nadawanie kolejnego bajtu. Jeżeli jednak nadane zostaną wszystkie bajty z bufora nadawczego, to ten automatyczny proces zostanie przerwany. Umieszczając kolejne bajty w buforze nadawczym należy zainicjować proces nadawania od początku. Wykonywane jest to przez ustawienie flagi TI, a tym samym programowe wygenerowanie przerwania. Dalej przebiegnie już wszystko automatycznie. Dla określenia, kiedy należy transmisję zainicjować, wprowadzono specjalną flagę: „NADAJE". Flaga ta jest ustawiana przy inicjalizacji nadawania i zerowania w momencie nadania ostatniego bajtu.
Przy transmisji, szczególnie na większą odległość, zdarzają się błędy. Należy się przed tym zabezpieczyć. Najprostszym takim zabezpieczeniem jest przesyłanie z każdym bajtem dodatkowego bitu parzystości. Kontrola poprawności tego bitu pozwala przeważnie na wyłapanie błędnie przesłanych bajtów.
W mikrokontrolerze 8051 można do tego typu transmisji wykorzystać tryb 3 sterownika transmisji. Tryb ten różni się od trybu 1 tylko tym, że po 8 bitach danych przesyłany jest dodatkowo 9 bit. Bit ten pobierany jest przez sterownik transmisji z rejestru SCON – bit TB8. Przez odpowiednie jego ustawienie decyduje się, który bit zostanie nadany.

Bit ten może być wykorzystany jako dodatkowy bit stopu (ustawiony stale na 1) lub jako bit kontroli parzystości, jeśli jego wartość będzie każdorazowo ustalana przy wysyłaniu bajtu. Kontrola parzystości może kontrolować parzystość bądź nieparzystość liczby bitów równych 1 w bajcie. Kontrola parzystości (even parity) polega na tym, że bit parzystości ustawiany jest tak, aby liczba jedynek w bajcie wraz z bitem parzystości była parzysta. NA przykład, przy przesyłaniu cyfr bit parzystości wygląda tak:

Znak	Bajt	Bit parzystości	Liczba jedynek
1	0011 0001	1	3 + 1
2	0011 0010	1	3 + 1
3	0011 0011	0	4 + 0
4	0011 0100	1	3 + 1

Kontrola nieparzystości wygląda przeciwnie.
W przykładzie 5 realizowana jest transmisja z bitami parzystości.

```
LED    EQU   P1.7

;************** Ustawienie TIMERów ************
;TIMER 0
T0_G  EQU   0                    ; GATE
T0_C  EQU   0                    ; COUNTER/-TIMER
T0_M  EQU   1                    ; MODE (0..3)
TIM0  EQU   T0_M+T0_C*4+T0_G*8
; TIMER 1
T1_G  EQU   0                    ;GATE
T1_C  EQU   0                    ; COUNTER/-TIMER
T1_M  EQU   0                    ; MODE (0..3)
TIM1  EQU   T1_M+T1_C*4+T1_G*8

TMOD_SET   EQU   TIM0+TIM1*16

;***************Transmisja szeregowa***************
TR_M EQU   1                     ; tryb transmisji (1...3)
TR_R  EQU   0                    ; zezwolenie na odbiór

SCON_SET   EQU   TR_M*64+TR_R*16

; SMOD=1
; TIMER1=57600/300bodów=192
TH1_SET      EQU   256-192
TL1_SET      EQU   256-192
;*********************************************

        LJMP  STAR
        ORG   100H
```

START:

```
        MOV  SCON, #SCON_SET        ; port szeregowy
        ORL  PCON, #80H             ; SMOD=1
        MOV  TMOD, #TMOD_SET        ; Timer 1 dla
        MOV  TH1, #TH1_SET          ; transmisji
        MOV  TL1, #TL1_SET
        SETB TR1                    ; start Timera 1

        SETB TI
        LCALL      WAIT_KEY
        JNB  TI, $                  ; czy można nadać
        CLR  TI                     ; kolejny znak
        ADD  A, #30H
        MOV  C, P                   ; ustaw bit parzystości
        MOV  RB8, C                 ; do nadania
        MOV  LED, C                 ; pokazanie bitu parzyst.
        MOV  SBUF, A                ; nadaj znak
        LCALL      WRITE_DATA       ; wyświetl na LCD
        SJMP LOOP
```

Liczenie liczby 1 w bajcie przed wysłaniem byłoby dość uciążliwe. Na szczęście w rejestrze stanu istnieje bit parzystości ‘P’. Bit parzystości jest ustawiany zgodnie z liczbą jedynek znajdujących się w akumulatorze. Przepisanie tego bitu do TB8, w momencie, gdy w akumulatorze znajduje się bajt do wysłania, powoduje wysłanie 9 bitu zgodnie z kontrolą parzystości. Ustawienie linii LED zgodnie z tym bitem pozwala na porównanie ustawienia tego bitu dla różnych znaków. Odpowiednie zanegowanie tego bitu pozwala na transmisję z kontrolą nieparzystości.

ZADANIA

ZADANIE 1

Ustawić inną (wybraną z typowych) prędkość transmisji w przykładzie 1, 2 lub 3.

ZADANIE 2

Zmodyfikować przykład 3 tak, aby nie było możliwości wpisania kolejnego bajtu do nadawania przed zakończeniem nadawania poprzedniego bajtu.

ZADANIE 3

Po uruchomieniu przykładu 4 we współpracy z komputerem można zaobserwować takie zjawisko: stałe naciśnięcie klawisza na klawiaturze komputera powoduje jednostajne dopisywanie znaków na wyświetlaczu LCD. Jeżeli jednak jednocześnie naciśnięty zostanie klawisz [Enter] na klawiaturze DSM-51, a tym samym rozpocznie się nadawanie do komputera, to dopisywanie na wyświetlacz LCD jest skokowe – po kilka znaków. Z czego to wynika?

Podstawy Techniki Mikroprocesorowej – studia dzienne i zaoczne
Technika Mikroprocesorowa – studia zaoczne magisterskie uzupełniające

ZADANIE 4

Zmodyfikować przykład 2 dodając kontrolę parzystości. Znaki odebrane prawidłowo wyświetlać na wyświetlaczu, a błędy w odbiorze sygnalizować na przykład diodą TEST.

WSKAZÓWKI

Ad. 1

Należy wpisać inne wartości do Timera 1. Wartości te mają być wyliczone zgodnie z podanym wzorem. Sprawdzić działanie ustawiając założoną prędkość na komputerze.

Ad. 2

Można to osiągnąć wprowadzając dodatkową flagę, analogicznie do flagi NADAJE w przykładzie 4. Jedno z możliwych rozwiązań przedstawia przykład 6 na dyskietce.

Ad. 3

Przy naciśniętym klawiszu [Enter] program główny „w kółko” dopisuje do bufora nadawczego ciąg znaków. Bufor bardzo szybko się zapełnia. Program główny czeka aż wpisze do bufora cały ciąg. W tym czasie napełniany jest bufor odbiorczy. Po wpisaniu ostatniego znaku z ciągu, program główny wybiera znaki z bufora odbiorczego, aż wyczyści cały bufor. Stąd dopisywanie na wyświetlacz odbywa się skokowo po kilka znaków.

Ad.4

Po odczytaniu bajtu z SBUF do akumulatora należy porównać flagi RB8 (odebrana parzystość) i P (faktyczna parzystość akumulatora). Prawidłowo działający program jest zamieszczony jako przykład 7.
Kontrolę działania tego programu można przeprowadzić nadając z komputera bez parzystość. Wtedy odebrany bit parzystości będzie zawsze 1 (nadany bit stopu). Dla niektórych znaków będzie to prawidłowe, dla innych nie.