

**Katedra Miernictwa Elektronicznego
Wydział Elektroniki, Telekomunikacji
i Informatyki
Politechniki Gdańskiej**



LABORATORIUM MIKROKONTROLERY I MIKROSYSTEMY

Ćwiczenie nr 5, 6 i 7

**Ćwiczenie 5: Programowanie i uruchamianie
oprogramowania w asemblerze na mikrokontroler 80C52**

**Ćwiczenie 6: Współpraca mikrokontrolera 80C52 z
zewnętrznymi urządzeniami peryferyjnymi**

**Ćwiczenie 7: Wykorzystanie języka wysokiego poziomu do
oprogramowania mikrokontrolera 80C52 na przykładzie
języka C**

dr inż. Zbigniew Czaja

Gdańsk 2003

Ćwiczenie nr 5

Programowanie i uruchamianie oprogramowania w asemblerze na mikrokontroler 80C52

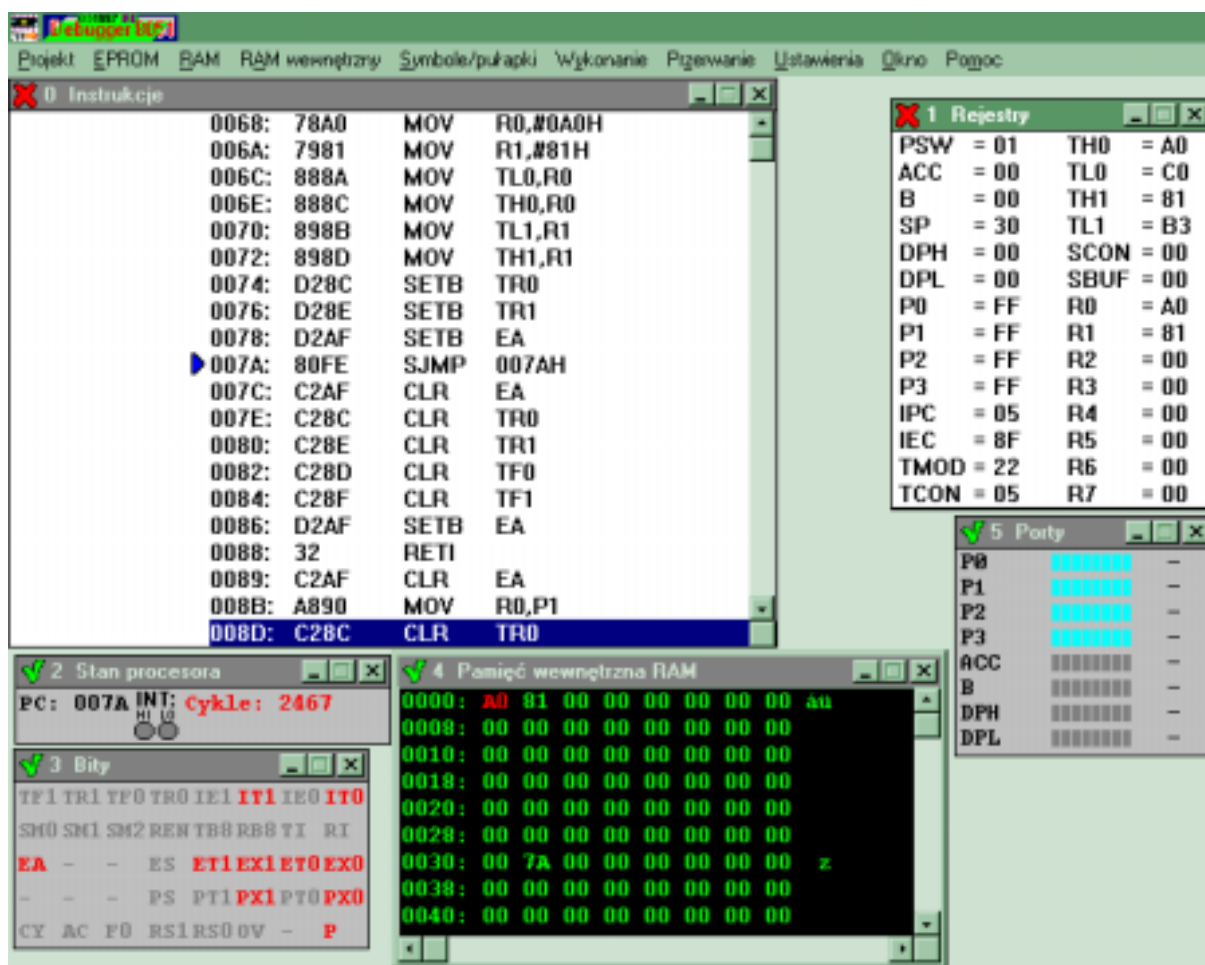
I. Sprzęt i oprogramowanie

- zestaw laboratoryjny mikrokontrolera 80C52 SW51EVB,
- emulator pamięci EPROM *picco-512*,
- zasilacz stabilizowany 9V,
- kabel połączeniowy dla standardu RS232 oraz przewody montażowe,
- środowisko programistyczne IDE (cross asembler firmy MetaLink **c:\labmik\lab5\asm51.exe**, oprogramowanie do obsługi emulatora sprzętowego **c:\labmik\piccolo.exe** (wersja przeznaczona do przetwarzania wsadowego **c:\labmik\picco512.exe**), symulator programowy **c:\labmik\dbg8051.exe**).

II. Zadania laboratoryjne

1. Przygotowanie stanowiska laboratoryjnego do pracy.
 - a) sprawdzić czy zestaw laboratoryjny SW51EVB jest podłączony do zasilania **9V**,
 - b) sprawdzić czy zestaw laboratoryjny jest podłączony przez kabel RS232 do portu COM1 komputera,
 - c) sprawdzić emulator pamięci EPROM *picco-512* jest podłączony do zestawu i do portu COM2 komputera,
połączyć (sprawdzić) przewodami montażowymi ze sobą następujące pary kołków (punktów testowych z próbnikami): TP1 z PR1, TP2 z PR2 i TP3 z RP3. Znaczenie zworek jest następujące: J1 2-3 zewnętrzna pamięć, J3 1-2 zerowanie po włączeniu zasilania (bez watchdog'a), J4 2-3 pamięć 27C64 lub 27C128,
 - d) uruchomić komputer (praca pod Windows 98),
 - e) przejść do katalogu **C:\labmik** i usunąć podkatalog **lab5** z całą jego zawartością,
 - f) rozpakować plik **lab5.zip** znajdujący się w katalogu **C:\labmik** (zawiera on podkatalog **lab5** wraz z jego właściwą zawartością).
2. Zapoznanie się ze środowiskiem mikrokontrolera 80C51/52 – kompilacja.
 - a) przejść do katalogu **C:\labmik\lab5** i skompilować program **przyk1.asm** z kodem źródłowym wydając polecenie: **asm51.exe przyk1.asm**,
 - b) przeanalizować powstały plik **przyk1.lst** zawierający listing programu przyk1.asm oraz obliczyć czas opóźnienia uzyskanego za pomocą podprogramu ZWLOKA,
3. Zapoznanie się ze środowiskiem mikrokontrolera 80C51/52 – symulacja. **Uwaga:** na potrzeby symulacji usunąć z pliku źródłowego przyk1.asm funkcję ZWLOKA i odwołania do niej (np. zapisując go pod nazwą **przyk1a.asm**) i ponownie skompilować nowo utworzony program.
 - c) uruchomić program **dbg8051.exe** znajdujący się w katalogu **C:\labmik**. Następnie załadować plik **przyk1a.hex** wydając polecenie z menu **EPROM ->Wczytaj Intel hex....** Pojawi się okienko, w którym wybrać plik **przyk1a.hex** i nacisnąć przycisk **OK**.

- d) Za pomocą poleceń z menu **Wykonanie** symulować pracę programu (zwłaszcza polecenie **Instrukcja** – klawisz **F7**). Korzystając z menu **Przerwanie** możliwe jest generowanie przerwań dostępnych w mikrokontrolerze 80C51.
- e) Wykonując polecenie **Instrukcja** należy pamiętać, iż wskaźnik aktualnie wykonywanej instrukcji (niebieski trójkącik) umieszczony po lewej stronie bieżącej instrukcji w okienku „Instrukcje” znajduje się pod adresem (pierwsza kolumna) wskazanym przez zawartość pola PC w okienku „Stan procesora”. Niestety zawartość okienka „Instrukcje” nie przesuwają się automatycznie za ruchem tego wskaźnika. Zatem aby móc obserwować, która instrukcja jest aktualnie wykonywana należy przesuwać zawartość okienka „Instrukcje” suwakami góra/dół, tak aby widzieć w nim wskaźnik aktualnie wykonywanej instrukcji.

Rys. 1. Główny ekran programu **dbg8051.exe**

- f) Na zakończenie symulacji opuścić program wydając polecenie **Projekt -> Koniec** lub naciskając klawisze **Alt-X**.
4. Zapoznanie się ze środowiskiem mikrokontrolera 80C51/52 – praca emulatora sprzętowego pamięci EPROM. Zasilanie emulatora doprowadzane jest z pakietu, na którym wykorzystywana jest symulowana pamięć, dlatego do poprawnej pracy konieczne jest włączenie zasilania pakietu uruchomieniowego.
- a) uruchomić program **piccolo.exe** (praca interaktywna), menu głównego wybrać **File-> Load**. Pojawi się okienko „File Load”. W polu „File name” ma być ***.hex**. Za pomocą klawisza **<Tab>** przejść do okienka „File list” i korzystając z klawiszy strzałek przejść

- do katalogu **c:\labmik\lab5** i wybrać plik **przyk1.hex**. Następnie wybrać **Simulator -> Transmit-> File** i właściwy plik (przyk1.hex), a potem **Simulator -> Simulation-> ON**. Dioda w symulatorze powinna świecić. Po przesłaniu pliku przyk1.hex do symulatora można opuścić program.
- b) nacisnąć klawisz RESET umieszczony na dodatkowej płytce zestawu laboratoryjnego, po czym prześledzić pracę programu (czy diody D4, D5 i D6 migają?).
5. Przeanalizowanie pozostałych programów znajdujących się w podkatalogu **lab5** (kompilacja, przeanalizowanie listingów, uruchomienie w systemie (można skorzystać z pracy wsadowej wywołując batch **p.bat** z parametrem będącym nazwą pliku heksalnego np. **p przyk2**) oraz analiza ich działania):
- a) **przyk2.asm** – program, w którym następuje przekazanie parametrów do podprogramu. W jaki sposób jest to realizowane?
- b) **przyk3.asm** – program sterujący pracą diod, lecz operujący na bajtach. Dlaczego użyto instrukcji ORL i ANL?
- c) **przyk4.asm** – program korzystający ze stosu. Kiedy zachodzi konieczność korzystania ze stosu?
- d) **przyk5.asm** – program prezentujący obsługę przerwań od licznika T0,
- e) **przyk6.asm** – program, w którym wykorzystano licznik T0 do precyzyjnego odmierzenia czasu. Jakie korzyści płyną ze stosowania przerwań w odmierzaniu precyzyjnych odcinków czasu i dlaczego?
- f) **przyk7.asm** – program przedstawiający obsługę przerwania zewnętrznego.
6. Napisać kod źródłowy własnego programu zgodnie z zaleceniami prowadzącego laboratorium. Następnie postępując jak opisano w punktach 2 i 4 (ewentualnie w 3) uruchomić swój własny projekt.

Uwaga: – dokumentacja do cross semblera używanego w ćwiczeniach 5 i 6 znajduje się w pliku **asm51.pdf** w katalogu **c:\labmik\doc5**,
– warto używać edytora tekstowego **ConTEXT**, który koloruje składnię.

Ćwiczenie nr 6

Współpraca mikrokontrolera 80C52 z zewnętrznymi urządzeniami peryferyjnymi

I. Sprzęt i oprogramowanie

- zestaw laboratoryjny mikrokontrolera 80C52 SW51EVB,
- emulator pamięci EPROM *picco-512*,
- zasilacz stabilizowany 9V,
- kabel połączeniowy dla standardu RS232 oraz przewody montażowe,
- środowisko programistyczne IDE (cross assembler firmy MetaLink **c:\labmik\lab6\asm51.exe**, oprogramowanie do obsługi emulatora sprzętowego **c:\labmik\piccolo.exe** (wersja przeznaczona do przetwarzania wsadowego **c:\labmik\picco512.exe**)).

II. Zadania laboratoryjne

1. Przygotowanie stanowiska laboratoryjnego do pracy.
 - a) sprawdzić czy zestaw laboratoryjny SW51EVB jest podłączony do zasilania **9V**,
 - b) sprawdzić czy zestaw laboratoryjny jest podłączony przez kabel RS232 do portu COM1 komputera,
 - c) sprawdzić emulator pamięci EPROM *picco-512* jest podłączony do zestawu i do portu COM2 komputera,
 - d) połączyć (sprawdzić) przewodami montażowymi ze sobą następujące pary kołków (punktów testowych z próbnikami): TP1 z PR1, TP2 z PR2 i TP3 z RP3.
 - e) uruchomić komputer (praca pod Windows 98),
 - f) przejść do katalogu **C:\labmik** i usunąć podkatalog **lab6** z całą jego zawartością,
 - g) rozpakować plik **lab6.zip** znajdujący się w katalogu **C:\labmik** (zawiera on podkatalog **lab6** wraz z jego **właściwą zawartością**).
2. Przeanalizowanie programów znajdujących się w podkatalogu **lab6** (kompilacja, przeanalizowanie listingów, uruchomienie w systemie oraz analiza ich działania – tak jak w ćwiczeniu 5):
 - a) **przyk1.asm** – prosty program sterujący wyświetlaczem LCD,
 - b) **przyk2.asm** – program obsługi klawiatury. Stan klawiszy odzwierciedlają 4 młodsze bity szyny danych odpowiednio MENU=>D0, Up=>D1, Down=>D2, Enter=>D3. Należy uruchomić poniższy program i obserwować reakcję na naciskanie klawiszy. Zmienić znaki przypisane klawiszom,
 - c) **przyk3.asm** – program sterujący pracą interfejsu UART. Do obsługi terminala na komputerze PC uruchomić program **terminal.exe** z katalogu **c:\labmik\terminal** (Aby transmisja przebiegała prawidłowo należy w okienku „Parametry transmisji” wywoływanym przez **Połączenie -> Linia** wprowadzić (o ile są inne) następujące wartości: Port: **Com2**, Bitów na sekundę: **1200**, Bity danych: **8**, Parzystość: **Brak**, Bity stopu: **1**, Kontrola transmisji: **Brak**. Warto włączyć echo **Połączenie -> Echo**. Na zakończenie należy otworzyć port wykonując polecenie **Połączenie -> Port otwarty**).

W poniższych przykładach należy przysyłać znaki z klawiatury PC na wyświetlacz pakietu oraz skromny zestaw symboli z klawiatury pakietu na ekran terminala. Proszę zmienić znaki przypisane klawiszom pakietu,

- d) **przyk4.asm** – program obsługi układu watchdog. Aby włączyć układ watchdog należy zworkę J3 przełożyć w pozycję 2-3. Należy zaobserwować działanie programu dla różnych położenia zworki J2 oraz odpowiedzieć na pytanie: jaki jest okres pobudzania watchdog'a, a następnie zmienić kod programu tak, aby jego działanie nie zależało od ustawień zworki. Na zakończenie pracy wrócić do ustawień początkowych zworek.
 - e) **przyk5.asm** – program obsługi zegara czasu rzeczywistego PCF8583 (interfejs I²C) i pamięci EEPROM ST93C46AB1 (interfejs Microwire).
3. Napisać kod źródłowy własnego programu zgodnie z zaleceniami prowadzącego laboratorium. Następnie uruchomić swój własny projekt.

Uwaga: warto używać edytora tekstowego **ConTEXT**, który koloruje składnię.

Ćwiczenie nr 7

Wykorzystanie języka wysokiego poziomu do oprogramowania mikrokontrolera 80C52 na przykładzie języka C

I. Sprzęt i oprogramowanie

- zestaw laboratoryjny mikrokontrolera 80C52 SW51EVB,
- emulator pamięci EPROM *picco-512*,
- zasilacz stabilizowany 9V,
- kabel połączeniowy dla standardu RS232 oraz przewody montażowe,
- środowisko programistyczne IDE (kompilator języka C **c:\labmik\c51\bin\c51.exe**, linker **c:\labmik\c51\bin\l51.exe**, konwerter plików na postać hexalną **c:\labmik\c51\bin\ohs51.exe**, oprogramowanie do obsługi emulatora sprzętowego **c:\labmik\piccolo.exe** (wersja przeznaczona do przetwarzania wsadowego **c:\labmik\picco512.exe**)).

II. Zadania laboratoryjne

1. Przygotowanie stanowiska laboratoryjnego do pracy.
 - a) sprawdzić czy zestaw laboratoryjny SW51EVB jest podłączony do zasilania 9V,
 - b) sprawdzić czy zestaw laboratoryjny jest podłączony przez kabel RS232 do portu COM1 komputera,
 - c) sprawdzić emulator pamięci EPROM *picco-512* jest podłączony do zestawu i do portu COM2 komputera,
 - d) połączyć (sprawdzić) przewodami montażowymi ze sobą następujące pary kołków (punktów testowych z próbnikami): TP1 z PR1, TP2 z PR2 i TP3 z RP3.
 - e) uruchomić komputer (praca pod Windows 98),
 - f) przejść do katalogu **C:\labmik** i usunąć podkatalog **lab7** z całą jego zawartością,
 - g) rozpakować plik **lab7.zip** znajdujący się w katalogu **C:\labmik** (zawiera on podkatalog **lab7** wraz z jego właściwą zawartością).
2. Kompilacja i uruchomienie **przyk1.c**:
 - a) Przeanalizować kod programu **przyk1.c** oraz pliku nagłówkowego **reg52.h** zawierającego wszystkie deklaracje rejestrów i peryferii z katalogu **c:\labmik\c51\inc**. Najwygodniej pliki te otwierać w edytorze tekstowym ConTEXT (koloruje składnie).
 - b) Dokonać kompilacji wywołując batch **c.bat** z nazwą pliku źródłowego bez rozszerzenia (zapis: **c przyk1**) – przeanalizować ten batch. Uzyskujemy w ten sposób dwa pliki wynikowe: relokowalny plik **przyk1.obj** oraz listing **przyk1.lst** zawierający informacje o procesie kompilacji wraz z kodem asemblera zastępującym kod w języku C. Przeanalizować plik **przyk1.lst**. Porównać pętlę opóźnienia wygenerowaną w ten sposób z pętlą wykorzystywaną w ćwiczeniu 5.
 - c) Dokonać kompilacji, linkowania i konwersji pliku **przyk1.c** do pliku **przyk1.hex** wydając polecenie **call przyk1**. (Batch **call.bat** wykonuje wszystkie powyższe czynności). Przeanalizować plik **call.bat**.
 - d) Załadować **plik1.hex** do pamięci EPROM *picco-512* podłączonego do zestawu laboratoryjnego (tak jak w ćwiczeniach 5 i 6) oraz zaobserwować pracę tego programu.

3. Skompilować **plik2.c** dla różnych modeli pamięci: COMPACT, LARGE i SMALL. W tym celu należy zmodyfikować batch **call.bat** dodając do linii wywołania kompilatora **c51.exe** odpowiednie dyrektywy. Opis dyrektyw zawarty jest w pliku **c51.pdf** (od strony 20), który znajduje się w katalogu **c:\labmik\doc7**. Porównać długości uzyskanych kodów (pliki *.hex) oraz sposób „przetłumaczenia” na język assemblera i reprezentacji typów danych (pliki *.lst). Programów nie należy uruchamiać.
4. Przeanalizowanie programów znajdujących się w podkatalogu **lab7** (kompilacja, przeanalizowanie listingów, uruchomienie w systemie oraz analiza ich działania – tak jak w podpunktach 2c i 2d):
 - a) **przyk3.c** – program do obsługi portu szeregowego. Aby nawiązać połączenie z zestawem laboratoryjnym należy na komputerze PC uruchomić emulator terminala np. **terminal.exe** znajdującym się w katalogu **c:\labmik\terminal**. Zaobserwować działanie programu. Przejrzeć kod programu oraz definicje podstawowych funkcji we/wy zawarte w pliku nagłówkowym **stdio.h**.
 - b) **przyk4.c** – program do obsługi wyświetlacza LCD. Po sprawdzeniu działania programu przeanalizować jego kod, a także kod źródłowy biblioteki obsługi wyświetlacza LCD **lcdlib.c**.
 - c) **przyk5.c** – program prezentuje wykorzystanie zewnętrznej pamięci RAM.
 - d) **przyk6.c** – program pokazuje jak obsługiwać przerwania w języku C. Po kompilacji przejrzeć listing programu i porównać wygenerowany kod w assemblerze z kodem zawartym w **przyk7.asm** z ćwiczenia 5.
5. Napisać kod źródłowy własnego programu zgodnie z zaleceniami prowadzącego laboratorium. Następnie uruchomić swój własny projekt. Na początku warto korzystać ze batcha **c.bat**, gdyż dokonuje on tylko kompilacji, a zatem przekazuje on głównie komunikaty o popełnionych błędach, co ułatwia usuwanie błędów z kodu źródłowego.

Uwaga: – dokumentacja do języka C znajduje się w katalogu **c:\labmik\doc7**,
– warto używać edytora tekstowego **ConTEXT**, który koloruje składnię.

III. Dokumentacja – opis mikrokontrolera 80C51/52

Uwaga: w instrukcji zawarto wyłącznie niezbędne informacje pomocne przy pisaniu oprogramowania. Szczegóły można znaleźć w odpowiednich dokumentach PDF i notach aplikacyjnych producentów mikrokontrolerów.

1. Wiadomości wstępne

Mikrokontrolery 8051/52 charakteryzują się następującymi właściwościami:

- 8-bitowy procesor
- wewnętrzny układ oscylatora
- 32 linie we/wy
- 4 kB wewnętrznej przestrzeni adresowej pamięci programu (8 kB w '32, '52)
- 64 kB zewnętrznej przestrzeni adresowej pamięci programu
- 64 kB zewnętrznej przestrzeni adresowej pamięci danych
- dwa 16-bitowe liczniki/zegary (trzy w układach '32, '52)
- pięć źródeł przerwań (sześć dla '52) z ustawianym priorytetem
- procesor boolowski umożliwiający operacje logiczne na pojedynczych bitach
- port szeregowy z pełnym duplexem (równoczesny odbiór i nadawanie)

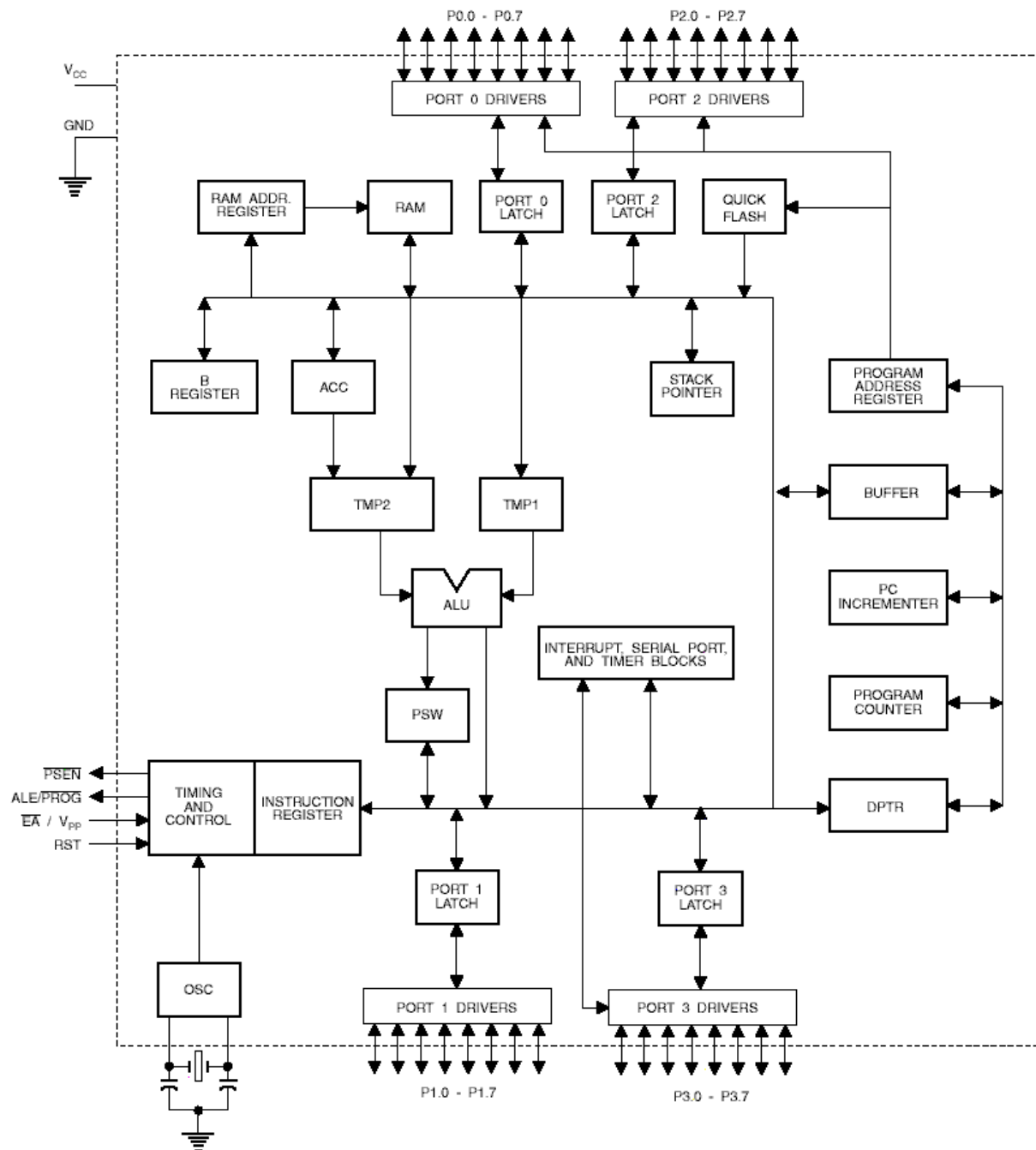
Mikrokontroler jest złożony z następujących elementów:

- generatora i układów kontrolno-taktujących
- pamięci programu ROM lub EPROM
- 128-bajtowej (256 w '32, '52) wewnętrznej pamięci danych RAM
- 16-bitowego licznika rozkazów (PC)
- 8-bitowej jednostki arytmetyczno-logicznej (ALU)
- czterech grup rejestrów, z których każda zawiera osiem 8-bitowych rejestrów umieszczonych w wewnętrznej pamięci RAM
- 16-bitowego wskaźnikowego rejestru danych (DPTR)
- 8-bitowego wskaźnika stosu (SP)
- zespołu rejestrów specjalnych (SFR) umieszczonych w wydzielonym obszarze pamięci wewnętrznej
- czterech 8-bitowych portów wejściowo-wyjściowych P0÷P3
- dwóch (trzech w '32, '52) układów czasowo-licznikowych: T0 i T1 (oraz T2 w '32/'52)
- układu szeregowej transmisji danych
- kontrolera priorytetu przerwań

Wiele wyprowadzeń kontrolera jest multipleksowanych:

- porty P0 i P2 adresują zewnętrzną pamięć kodu programu lub danych
- port P0 służy do przesyłania 8-bitowych danych
- port P3 pełni rolę wejść-wyjść układów licznikowo-czasowych i portu szeregowego oraz steruje kierunkiem przepływu danych do/z zewnętrznej pamięci danych RAM.

Wprowadza to pewne ograniczenia, np. nie można korzystać z portów P0 i P2 przy zewnętrznej pamięci programu lub danych.



Rys. 1. Schemat blokowy mk 80C51/52

Z ALU są funkcjonalnie powiązane trzy rejestry o ściśle sprecyzowanym przeznaczeniu:

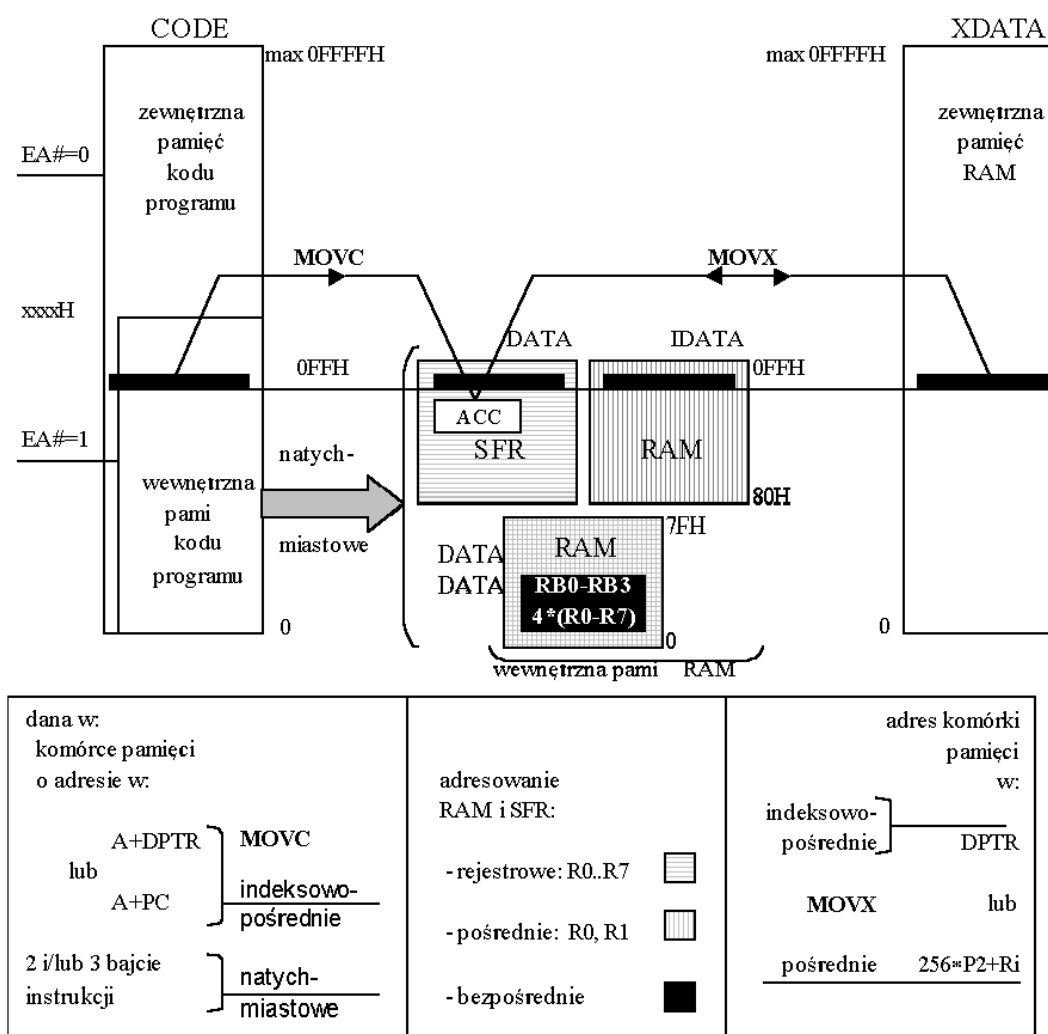
- *akumulator A* (inaczej ACC) służy do przechowywania jednego z argumentów w operacjach arytmetycznych i logicznych oraz do przechowywania wyniku tych operacji. Jednak część operacji logicznych jest wykonywanych bez użycia akumulatora. Tylko w akumulatorze można przesunąć daną o jeden bit w prawo lub w lewo. Akumulator uczestniczy również przy przesyłaniu danych do i z pamięci zewnętrznej RAM oraz pobierania argumentów z pamięci kodu programu.
- rezultaty operacji wykonywanych w ALU są zapamiętywane w *rejestrze słowa statusowego PSW* (program status word). Wybrane bity tego rejestru informują o wyniku wykonywanych działań:

- przekroczeniu zakresu liczb całkowitych ze znakiem i bez znaku w operacjach dodawania i odejmowania
- korekcji sumy liczb przedstawionych w kodzie BCD
- próbie dzielenia przez zero
- parzystości stanu akumulatora
- wyborze grupy dostępnych rejestrów wewnętrznych
- w operacjach mnożenia i dzielenia jeden z argumentów przechowywany jest w akumulatorze, a drugi w *rejestrze B*. Jeśli nie przewiduje się użycia tych operacji, to rejestr B może być wykorzystany jako dodatkowa komórka pamięci RAM.

Do adresowania pamięci kodu programu ROM i zewnętrznej pamięci RAM wykorzystywany jest wskaźnikowy rejestr danych DPTR (data pointer register).

Wszystkie rejestry procesora, poza licznikiem programów PC i rejestrze DPTR są rejestrami 8-bitowymi. Rejestry specjalne mają szczególne znaczenie. Każdy element struktury wewnętrznej ma swoją reprezentację w postaci 1-bajtowego lub 2-bajtowego rejestru. Wpisanie informacji do tych rejestrów inicjuje działanie poszczególnych bloków funkcjonalnych mikrokontrolera.

2. Organizacja pamięci mikrokontrolera



Rys. 2. Organizacja wewnętrznej i zewnętrznych pamięci mk 80C51/52

Na rys. 2 przedstawiono organizację zewnętrznych i wewnętrznych pamięci mikrokontrolera. Pamięć kodu programu oznacza się symbolem CODE, wewnętrzną pamięć RAM symbolem DATA lub IDATA w zależności od sposobu adresowania, a zewnętrzną pamięć RAM symbolem XDATA.

Rodzina mikrokontrolerów '51 może korzystać z następujących typów pamięci:

- wewnętrznej pamięci kodu programu typu ROM lub EPROM; mikrokontrolery 8031/32 nie posiadają wewnętrznej pamięci kodu programu
- zewnętrznej pamięci kodu programu dołączanej indywidualnie przez użytkownika
- wewnętrznej pamięci danych typu RAM
- zewnętrznej pamięci danych typu RAM
- zespołu rejestrów specjalnych SFR

Adresy wewnętrznej pamięci RAM pokrywają się z adresami zewnętrznej pamięci RAM i pamięci kodu programu (architektura harwardzka). Mikrokontroler rozróżnia typy pamięci dzięki stosowaniu odpowiednich rozkazów:

1. rozkazy dotyczące wewnętrznej pamięci RAM mają postać MOV...
2. rozkazy związane z zewnętrzną pamięcią RAM - postać MOVB...
3. rozkazy związane z pamięcią kodu programu - postać MOVC...

W mikrokontrolerach zawierających 256 bajtów pamięci wewnętrznej adresy górnej części wewnętrznej pamięci RAM pokrywają się z adresami obszaru rejestrów specjalnych. Rozróżnienie tych struktur odbywa się poprzez odpowiedni tryb adresowania.

2.1 Pamięć kodu programu

Obszar pamięci programu o maksymalnej objętości 64 kB może składać się z dwóch części:

1. pamięci wewnętrznej (ROM lub EPROM) o pojemności 4 kB o adresach 0 - 0FFFH lub 8 kB o adresach 0 - 1FFFH (w mikrokontrolerze 8052/32); na rys. 2 symbol XXXX oznacza adres 0FFFH lub 1FFFH
2. pamięci zewnętrznej, która może mieścić się w obszarze 0 - 0FFFH, a tym samym pokrywa obszar wewnętrznej pamięci.

Jeśli mikrokontroler posiada zewnętrzną pamięć ROM lub EPROM to zmieniając stan linii EA (external access enable) można wybierać między pamięcią wewnętrzną a zewnętrzną. Jeżeli wejście EA jest w stanie:

1. jedynki logicznej i adres komórki pamięci programu mieści się w obszarze wewnętrznej pamięci programu, to rozkaz jest pobierany z wewnętrznej pamięci ROM
2. jedynki logicznej i adres komórki pamięci programu przekracza obszar wewnętrznej pamięci programu to rozkazy są automatycznie pobierane z pamięci zewnętrznej
3. zera logicznego to rozkazy są pobierane wyłącznie z pamięci zewnętrznej. Dla procesorów 8031/32, ze względu na brak wewnętrznej pamięci programu, wejście EA musi być zawsze w stanie zera logicznego. Przy pobieraniu rozkazów i danych z zewnętrznej pamięci programu pojawia się sygnał PSEN (program strob enable), który uaktywnia pamięć kodu programu.

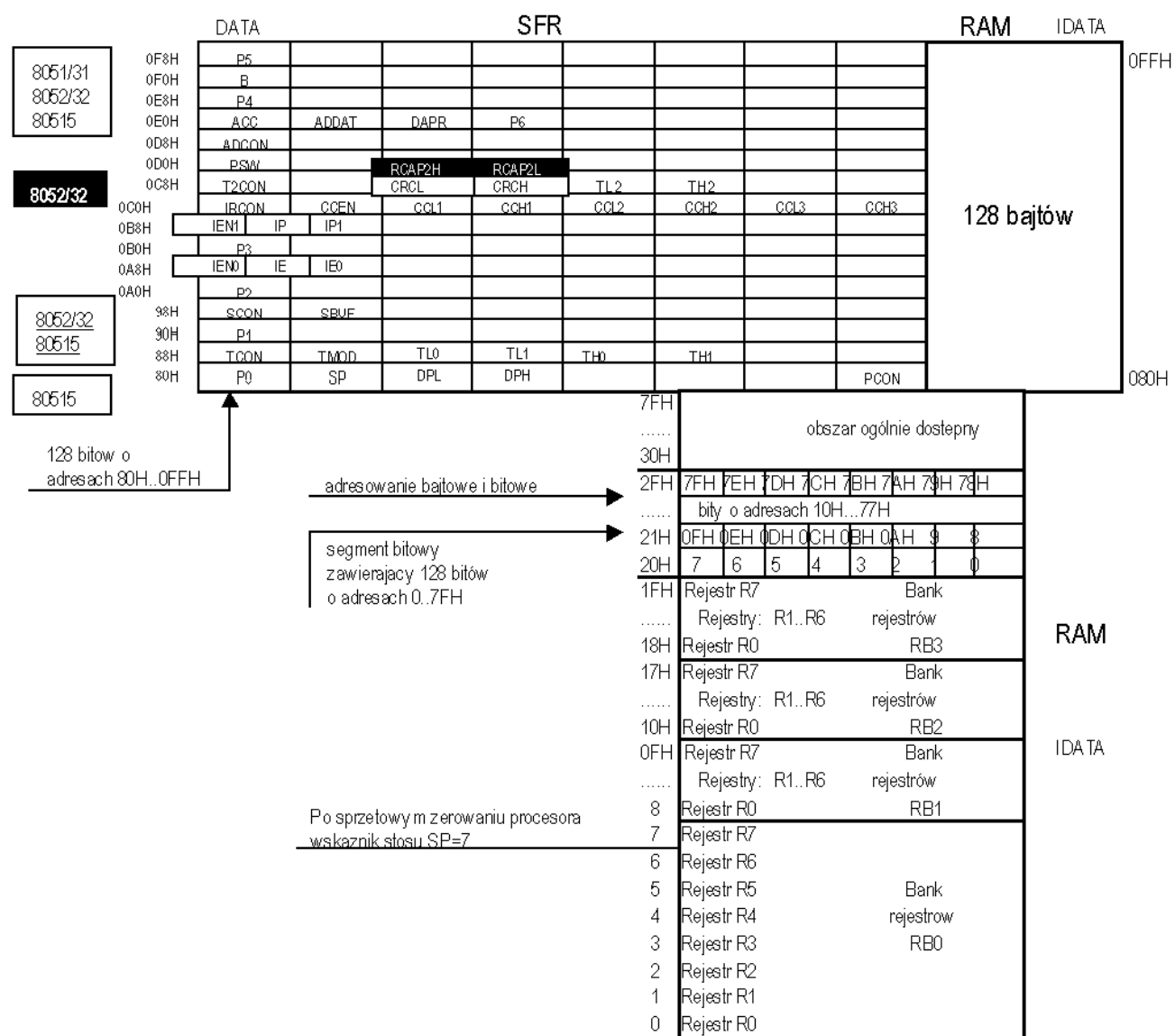
Tworząc program działania mikrokontrolera nie można zapominać, że niektóre adresy są zarezerwowane. Rezerwacja ta jest wynikiem szczególnych zachowań mikrokontrolera w trakcie jego startu oraz reakcji na wewnętrzne lub zewnętrzne przerwania. Wartości tych adresów przedstawiono w tabeli.

Źródła sygnału zerowania/przerwania	Adres
Start procesora	0000H
Przerwanie zewnętrzne INT0#	0003H
Przerwanie od licznika T0	000BH
Przerwanie zewnętrzne INT1#	0013H
Przerwanie od licznika T1	001BH
Przerwanie od portu szeregowego	0023H
Przerwanie od licznika T2 (8052/32)	002BH

Podane adresy przerwań wynikają z konstrukcji mikrokontrolera i nie mogą być zmieniane.

Po włączeniu zasilania mikrokontrolera lub po jego zerowaniu linią RESET zerowany jest również licznik rozkazów PC. Dlatego program procesora zaczyna się zawsze od adresu zerowego, pod którym znajduje się najczęściej rozkaz skoku do zasadniczego programu.

2.2 Wewnętrzna pamięć RAM



Rys. 3. Struktura wewnętrznej pamięci mk 80C51/52

Wewnętrzna pamięć RAM, oznaczona na rys. 3 symbolem IDATA, złożona jest z dwóch segmentów (lub trzech w mikrokontrolerze 8052/32). Jej zadaniem jest przechowywanie danych, wyników obliczeń, rezultatów działania podprogramów itp. Każdy segment pamięci zawiera 128 bajtów. Wewnętrzna pamięć RAM o adresach 0..7FH (0..127D) ma taką samą strukturę we wszystkich omawianych typach mikroprocesorów. Natomiast pamięć RAM o adresach 80H..0FFH (128..255D) złożona jest z jednego lub dwóch segmentów. Jednym z nich są zawsze rejestry specjalne, a drugim tylko w mikrokontrolerze 8052/32 może być pamięć ogólnego przeznaczenia. Rozróżnienie obu segmentów pamięci dokonywane jest za pomocą zmiany trybu adresowania. Adresowanie rejestrów specjalnych możliwe jest jedynie bezpośrednio, a pamięci RAM ogólnego przeznaczenia tylko pośrednio.

2.2.1 Wewnętrzna pamięć RAM - adresy 0..7FH

W tym 128-bajtowym fragmencie pamięci RAM można wyróżnić następujące obszary:

- cztery banki rejestrów ogólnego przeznaczenia oznaczonych symbolami RB0..RB3, adresy 0..1FH

Każdy z banków zawiera 8 rejestrów oznaczonych symbolami R0..R7. Łącznie w procesorze występują 32 rejestry, ale programowy dostęp możliwy jest tylko do jednego wybranego banku, czyli 8 rejestrów. W rejestrze słowa statusowego PSW (rejestr specjalny) dwa bity, RS1 i RS0, określają numer aktywnego banku rejestrów:

rejestr PSW				adres 0D0H			
CY	AC	F0	RS1	RS0	OV	F1	P

Zmiana aktywnego banku rejestrów możliwa jest np. przez użycie instrukcji ustawiającej - wpisując zero logiczne na pozycję wyróżnionych bitów RS0 i RS1.

Zerowanie procesora linią RST (RESET) wymusza także zerowanie rejestru słowa statusowego PSW. W efekcie wybierany jako aktywny jest zerowy bank rejestrów RB0.

Rejestry R0..R7 używane są do adresowania wewnętrznej pamięci RAM w trybie adresowania rejestrowego. W trybie adresowania rejestrowego pośredniego wykorzystywane są jedynie rejestry R0 i R1 pełniące rolę wskaźników lub rejestrów indeksowych. Taki sposób adresowania zapewnia dostęp zarówno do wewnętrznej jak i zewnętrznej pamięci RAM.

W trakcie zerowania procesora następuje wpisanie do wskaźnika stosu SP wartości 7. Efektem takiego działania jest wpisywanie danych od adresu 8H, rezerwowanego dla pierwszego banku rejestrów RB1. Jeśli użytkownik planuje wykorzystanie tego banku rejestrów to należy inicjalizować stos np. w przestrzeni adresowej 30H..7FH.

Rejestr wskaźnika stosu SP dostępny jest w przestrzeni rejestrów specjalnych, adres 81H.

- obszar o adresach 20H..2FH adresowany bajtowo i bitowo.
Adresy bajtów podane są na rys. 3 szesnastkowo. Adresy bitowe podane są we wnętrzu każdego bitu. Dostęp do każdego bitu możliwy jest na dwa sposoby:
 1. przez podanie adresu bitu z przedziału 0H..7FH
 2. przez odwołanie bajtowe i wskazanie numeru bitu w bajcie, np. bitu od 0 do 7 adresowane są jako 20.0..20.7, od 8 do 0FH jak 21.0..21.7 itd.
 Na rys. 3 wyróżniono jeden bit o adresie bajtowym i bitowym równym 24H. Ustawienie tego bitu możliwe jest w następujący sposób:

lub	SETB 24H	;bitowo, ustaw bit 24H
lub	SETB 24H.4	;bitowo, ustaw bit 4 w bajcie o adresie 24H
	MOV 24H,#10H	;bajtowo

- obszar 30H..7FH przewidziany jest do zastosowań ogólnych. Dostęp do tego obszaru możliwy jest jedynie bajtowo.

2.2.2 Wewnętrzna pamięć RAM - adresy 80H..0FFH

Obszar wewnętrznej pamięci RAM o adresach 80H..0FFH zorganizowany jest różnie w zależności od typu mikrokontrolera. W mikrokontrolerach 8051/31 dostępny jest jeden segment 128-bajtowej pamięci, a w mikrokontrolerach 8052/32 i nowszych dostępne są dwa segmenty pamięci. Oba rozwiązania przedstawiono na rysunku 3. Rozróżnienia obu segmentów pamięci dokonywane jest przez zmianę trybu adresowania. Segment zawierający rejestry specjalne dostępny jest tylko dzięki adresowaniu bezpośredniemu (segment oznaczony jako DATA), a segment pamięci RAM ogólnego przeznaczenia można adresować tylko w sposób pośredni (segment oznaczony jako IDATA). Pamięć wewnętrzna RAM o adresach 0..7FH adresowana może być zarówno bezpośrednio jak i pośrednio (segment oznaczony jako DATA lub IDATA).

Poniżej przedstawiono mapę wewnętrznej pamięci mikrokontrolera zawierającą rejestry specjalne. Wszystkie rejestry specjalne SFR adresowane są bajtowo, a rejestry wyróżnione na rysunku mogą być również adresowane bitowo. Adresy tych rejestrów są wielokrotnością 8 tzn. 80H, 88H, 90H, 98H, ..., 0F0H, 0F8H. Taki sposób adresowania zwiększa efektywność programu, zwłaszcza w wykorzystaniu niektórych rejestrów takich jak: porty P0..P3, akumulator ACC, rejestr statusowy PSW, rejestry kontrolujące pracę liczników i modułu transmisji szeregowej oraz rejestry struktury przerwań. Przykładowo bitowy adres 80H odpowiada najmniej znaczącemu bitowi portu P0, bitowy adres 8FH odpowiada najbardziej znaczącemu bitowi rejestru sterującego pracą zegara TCON.

0F8H								
0F0H	B							
0E8H								
0E0H	ACC							
0D8H								
0D0H	PSW							
0C8H	T2CON		RCAP2L	RCAP2H	TL2	TH2		
0C0H								
0B8H	IP							
0B0H	P3							
0A8H	IE							
0A0H	P2							
98H	SCON	SBUF						
90H	P1							
88H	TCON	TMOD	TL0	TL1	TH0	TH1		
80H	P0	SP	DPL	DPH				PCON

Jak widać, nie wszystkie rejestry są zagospodarowane, np. o adresach 0D1H..0DFH. Wolne rejestry nie mogą być jednak wykorzystane w programach jako dodatkowe komórki wewnętrznej pamięci RAM. Tylko niektóre z rejestrów mogą pełnić taką funkcję. O tym decyduje producent mikrokontrolera oraz programista.

Poniżej omówiono wybrane rejestry specjalne mikrokontrolera:

- Akumulator A, adres 0E0H.
- Rejestr B, adres 0F0H.
- Rejestr słowa statusowego PSW, adres 0D0H.

rejestr PSW				adres 0D0H			
CY	AC	F0	RS1	RS0	OV	F1	P

Znaczenie poszczególnych bitów jest następujące:

- CY (carry flag) - znacznik przeniesienia z bitu A7 akumulatora
- AC (auxiliary carry flag) - znacznik przeniesienia połówkowego między bitami A3 i A4 akumulatora
- F0 znacznik ogólnego przeznaczenia
- RS1 (register bank select 1) - bit 1 wyboru banku rejestrów - por. p. 2.2.1
- RS0 (register bank select 0) - bit 0 wyboru banku rejestrów
- OV (overflow flag) - znacznik nadmiaru
- F1 znacznik ogólnego przeznaczenia, brak w mikrokontrolerze 8051/31
- P (parity flag) - znacznik parzystości, czyli dopełnienie do parzystej liczby jedynek w akumulatorze

Znaczniki F0 i F1 są testowane w rozkazach skoków warunkowych.

Bit y RS1 i RS0 wyboru banku rejestrów umożliwiają uaktywnienie jednego z czterech banków:

RS1	RS0	Numer wybranego banku rejestrów	Adres wybranego banku
0	0	0	00H..07H
0	1	1	08H..0FH
1	0	2	10H..17H
1	1	3	18H..1FH

- Wskaźnik stosu SP (stack pointer), adres 81H
Wskazuje on ostatnią zajętą komórkę stosu. Jest modyfikowany automatycznie przy zapisie na stos i odczycie ze stosu: inkrementowany przed wykonaniem instrukcji PUSH (o jeden) i CALL (o dwa) oraz dekrementowany w trakcie wykonywania instrukcji POP (o jeden) i RET lub RETI (o dwa).
Po zerowaniu linią RST ustawiany jest SP=7.
Stos narasta w górę pamięci.
- Wskaźnikowy rejestr danych DPTR (data pointer), adresy 82H, 83H
16-bitowy rejestr złożony z dwóch części: starszej DPH (adres 83H) i młodszej (adres 82H). Jest on stosowany do adresowania 64 kB pamięci zewnętrznej RAM w trybie indeksowo-rejestrowym:

MOVX A,@DPTR

lub do pobierania argumentów z pamięci kodu programu:

MOVC A,@A + DPTR

2.3 Zewnętrzna pamięć RAM mikrokontrolera

Zewnętrzna pamięć RAM mikroprocesora przeznaczona jest do przechowywania dużych tablic danych lub zmiennych. Maksymalny rozmiar tej pamięci ograniczony jest, podobnie jak i pamięci kodu programu, do 64 kB. W obu typach pamięci wymagany jest 16-bitowy adres komórek pamięci. Zarówno licznik rozkazów PC i wskaźnikowy rejestr danych DPTR są rejestrami 16-bitowymi.

Zewnętrzną pamięć RAM można również adresować za pośrednictwem dwóch 8-bitowych rejestrów:

- portu P2 (8 starszych bitów adresu Adr₁₅..Adr₈)
- rejestru Ri, i=0 lub 1 (8 młodszych bitów adresu Adr₇..Adr₀)

Zawartość portu P2 określa numer segmentu (strony), a rejestr Ri adres komórki pamięci w obrębie strony (offset).

Dwie linie portu P3 pełnią rolę linii sterujących przesyłem danych:

- P3.6 = WR# , steruje wpisem danych do zewnętrznej pamięci RAM
- P3.7 = RD# , steruje odczytem danych z zewnętrznej pamięci RAM

Dwa przykłady współpracy z zewnętrzną pamięcią RAM:

1. Adresowanie za pomocą 16-bitowego rejestru DPTR:

```
MOV DPTR,#5A4EH      ;DPH←5AH
                      ;DPL←4EH
MOVX A,@DPTR          ;A←zawartość pamięci zewn. o adresie
                      ;zapisanym w DPTR
```

W czasie wykonywania instrukcji przesłania do akumulatora, do portu P2 wystawiana jest wartość rejestru DPH, do portu P0 wystawiana jest wartość rejestru DPL. W ten sposób przekazywany jest na zewnątrz kontrolera pełny, 16-bitowy adres komórki pamięci.

2. Adresowanie za pomocą portu i 8-bitowego rejestru.

```
MOV P2,#5AH           ;P2←adres strony pamięci zewnętrznej
MOV R1,#4EH           ;R1←offset adresu
MOVX A,@R1            ;A←(5A00H + 4EH)
```

W czasie wykonywania ostatniej instrukcji stan portu P2 nie ulega zmianie i zawiera starszą część adresu, zaś do portu P0 wystawiana jest zawartość rejestru R1, młodsza część adresu.

2.4 Tryby adresowania

W procesorach rodziny '51 wyróżnić można 5 trybów adresowania pamięci:

- za pomocą rejestrów, R (register), adresem argumentu jest numer rejestru,
- bezpośrednio, B (direct), adresem argumentu jest drugi lub trzeci bajt instrukcji,
- natychmiastowo, N (immediate), argument jest drugim lub trzecim bajtem instrukcji,
- rejestrowo pośrednio, P (register indirect), adresem argumentu jest zawartość wskazanego rejestru,

- za pomocą rejestrów bazowych, indeksowo-rejestrowo pośrednio, I (base register plus index-register indirect), adres argumentu zawarty w akumulatorze A i liczniku rozkazów PC lub wskaźnikowym rejestrze danych DPTR,
- adresowanie bitowe.

Adresowanie *za pomocą rejestrów (R)* realizowane jest przez użycie dowolnego rejestru R0..R7 z wybranego banku rejestrów RB0..RB3, akumulatora ACC, rejestru B, rejestru DPTR lub znacznika przeniesienia C. Najmniej znaczące bity kodu operacyjnego instrukcji określają, który z rejestrów jest wykorzystywany. Przykładem jest instrukcja przepisania:

```
MOV A,R3
```

Powyższa instrukcja posiada binarny kod 11101**011**B, gdzie wyróżniona sekwencja **011**B jest numerem rejestru R3.

Adresowanie *bezpośrednie (B)* jest jedynym sposobem adresowania komórek rejestrów specjalnych. Ten tryb może być także stosowany do adresowania 128 bajtów wewnętrznej pamięci RAM o adresach 0..7FH. Przykładem adresowania bezpośredniego jest instrukcja sumy logicznej akumulatora z zawartością komórki o adresie 0E0H:

```
ORL A,0E0H
```

Adresowanie *natychmiastowe (N)* wykorzystywane jest do pobierania argumentów z pamięci kodu programu, np.:

```
MOV R6,#5CH    ;wpisz do rejestru R6 wartość 5CH z pamięci kodu
                ;programu ROM/EPROM,
                ;wartość 5CH jest drugim bajtem instrukcji
```

W adresowaniu *rejestrowym pośrednim (P)* używane są jedynie rejestry R0 i R1 w ramach wybranego banku rejestrów. Oba 8-bitowe rejestry pełnią rolę wskaźników przy adresowaniu 256-bajtowego bloku pamięci, zarówno wewnętrznej pamięci RAM jak i 256-bajtowej strony zewnętrznej pamięci RAM. Ten sposób adresowania nie umożliwia adresowania rejestrów specjalnych mikroprocesora.

Adresowanie *za pomocą rejestrów bazowych, indeksowo-rejestrowe pośrednie (I)* umożliwia pobieranie danych z pamięci kodu programu i zewnętrznej pamięci RAM. Adres komórki pamięci jest złożeniem adresu zawartego we wskaźnikowym rejestrze danych DPTR lub liczniku rozkazów PC i rejestrze indeksowym, którego rolę pełni akumulator A:

```
MOVC A,@A+DPTR
```

3. Porty

Wszystkie cztery porty są dwukierunkowe. Porty P0, P2 i P3 są portami wielofunkcyjnymi.

Port P0, przy współpracy mikrokontrolera z zewnętrzną pamięcią programu jest wykorzystywany do wysyłania mniej znaczącego bajtu adresu rozkazu. Przy współpracy z zewnętrzną pamięcią danych przez port P0 jest również przesyłany mniej znaczący bajt adresu oraz są przesyłane dane do i z zewnętrznej pamięci RAM.

Na wyjściu portu P2, przy współpracy mikrokontrolera z zewnętrzną pamięcią programu lub danych, jest wysyłany bardziej znaczący bajt adresu.

W porcie P3 każde wyprowadzenie ma swoją alternatywną funkcję:

```
P3.0  RxD  - wejście portu szeregowego
P3.1  TxD  - wyjście portu szeregowego
```

- P3.2 INT0 - wejście przerwania zewnętrznego lub bramkowania licznika T0
 P3.3 INT1 - wejście przerwania zewnętrznego lub bramkowania licznika T1
 P3.4 T0 - wejście taktujące licznik T0
 P3.5 T1 - wejście taktujące licznik T1
 P3.6 WR - wyjście sygnału zapisu do zewnętrznej pamięci RAM
 P3.7 RD - wyjście sygnału odczytu z zewnętrznej pamięci RAM

Wpisanie danej do rejestru portu lub wykonanie operacji na zawartości rejestru powoduje wysłanie zawartości rejestru na końcówki portu, o ile port nie wykonuje funkcji alternatywnej.

Wykorzystanie funkcji alternatywnej wymaga wpisania wartości 1 do odpowiedniego bitu rejestru portu.

Bity w rejestrach portów posiadają swój własny adres i mogą być modyfikowane niezależnie. Poniżej są podane adresy bajtowe, pod którymi znajdują się rejestry portów oraz adresy bitowe poszczególnych portów.

P0	87H	86H	85H	84H	83H	82H	81H	80H	adres 80H
P1	97H	96H	95H	94H	93H	92H	91H	90H	adres 90H
P2	0A7H	0A6H	0A5H	0A4H	0A3H	0A2H	0A1H	0A0H	adres 0A0H
P3	0B7H	0B6H	0B5H	0B4H	0B3H	0B2H	0B1H	0B0H	adres 0B0H

Jak widać z powyższego zestawienia adresy bitów dla poszczególnych rejestrów rozpoczynają się od adresu odpowiadającego adresowi rejestru.

4. Liczniki T0 i T1

Mikrokontroler 8051 jest wyposażony w dwa 16-bitowe liczniki T0 i T1, z których każdy składa się z dwóch rejestrów TH0-TL0 i TH1-TL1. Mogą one pracować jako układy zliczające impulsy zewnętrzne (counter) lub impulsy zegara wewnętrznego (timer). Liczniki liczą w przód, mają cztery tryby pracy. Pierwsze trzy tryby pracy: 0, 1 i 2 są wspólne dla obydwu liczników. Do obsługi tych liczników są przeznaczone dwa rejestry: TCON i TMOD.

Rejestr TCON mieści się pod adresem 88H w SFR i zawiera znaczniki:

TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	adres 88H
------	-----	-----	-----	-----	-----	-----	-----	-----	-----------

TFn	przerzutnik przepełnienia licznika Tn (n=0,1), ustawiany, gdy nastąpi przepełnienie licznika, a zerowany po wejściu do obsługi przerwania od licznika Tn
TRn	znacznik sterujący pracą licznika Tn: TRn = 1 : licznik pracuje, TRn = 0 : praca licznika zatrzymana. Stan znacznika jest zmieniany programowo.
	Pozostałe znaczniki są związane z przerwaniami zewnętrznymi.

Każdy ze znaczników rejestru TCOM może być zmieniany indywidualnie.

Tryby pracy liczników, źródło sygnałów wejściowych oraz sposób sterowania licznikiem ustala się przez wpisanie do rejestru TMOD, znajdującego się pod adresem 89H w przestrzeni SFR, odpowiedniego słowa. W rejestrze tym nie można zmieniać pojedynczych bitów.

Rejestr TMOD zawiera następujące znaczniki:

GATE	C/T	M1	M0	GATE	C/T	M1	M0
↑↑				↑↑			
licznik 1				licznik 0			

Znaczenie znaczników jest następujące ($n=0,1$):

GATEn	
0	licznik Tn zlicza impulsy tylko wtedy, gdy znacznik TRn jest w stanie 1
1	licznik Tn zlicza impulsy tylko wtedy, gdy znacznik TRn jest w stanie 1 i wejście P3.2 lub P3.3 jest w stanie 1

C/Tn	
0	licznik Tn zlicza impulsy z zegara wewnętrznego
1	licznik Tn zlicza impulsy podane na wejście Tn (P3.4 lub P3.5)

M1	M2	określenie trybu liczników
0	0	tryb 0
0	1	tryb 1
1	0	tryb 2
1	1	tryb 3 - różne zachowanie liczników T1 i T0

4.1 Tryby pracy liczników

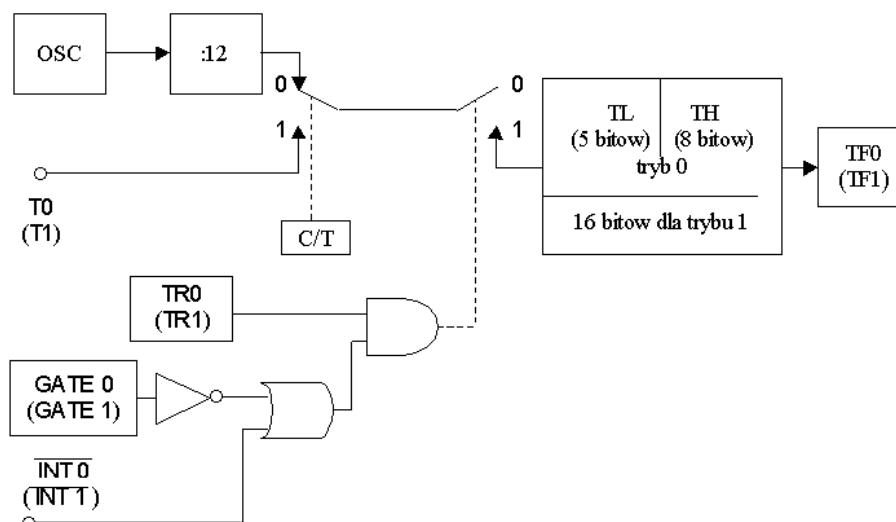
4.1.1 Tryb 0

W tym trybie (rys. 4) rejestr LTn licznika n ($n=0,1$) może pracować w układzie dzielnika wstępnego dzielącego przez 2^5 lub 5-bitowego dzielnika, natomiast rejestr THn licznika n - w układzie licznika 8-bitowego.

Ustawiając odpowiednie znaczniki C/T, TR i GATE można zmieniać konfigurację układu pracy licznika. Przy ustawieniu znacznika C/T = 0 wejście licznika jest dołączone do wewnętrznego zegara z sygnałem wyjściowym o częstotliwości $f_{osc}/12$. Przy C/T = 1 sygnał do licznika jest doprowadzany z odpowiedniego wejścia zewnętrznego. Znacznik TR służy do programowego uruchamiania pracy licznika. Przy TR = 0 licznik jest zatrzymany, a przy TR = 1 pracuje. Praca licznika może być również uruchamiana sygnałem zewnętrznym. Przy ustawieniu znacznika GATEn = 1 licznik pracuje gdy wejście INTn (P3.2 lub P3.3) jest na wysokim poziomie, jest natomiast zatrzymana, gdy wejście to jest na poziomie niskim. Gdy znacznik GATEn = 0 praca licznika nie zależy od stanu wejścia INTn. Przepelnienie licznika powoduje wpisanie stanu 1 do przerzutnika TFn. Jeżeli przerwanie od licznika jest aktywne, to nastąpi wywołanie procedury obsługi tego przerwania.

4.1.2 Tryb 1

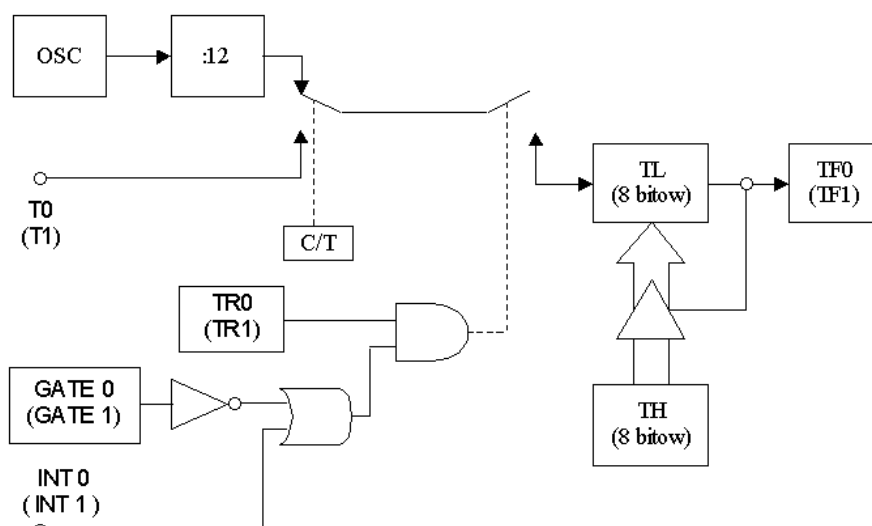
Tryb 1 jest taki sam jak tryb 0 z tą różnicą, że rejestr TLn pracuje jako licznik 8-bitowy. W tym trybie liczniki pracują jako liczniki 16-bitowe.



Rys. 4. Tryb 0 i 1 dla liczników T0 i T1 mk 80C51/52

4.1.3 Tryb 2

W trybie 2 licznik pracuje jako licznik 8-bitowy z ustawianym stanem początkowym - autoładowaniem, rys. 5.



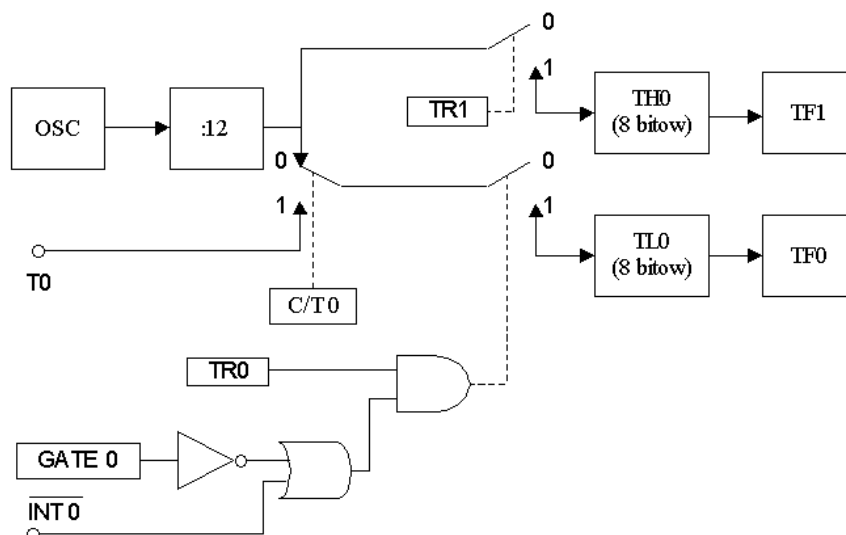
Rys. 5. Tryb 2 dla liczników T0 i T1 mk 80C51/52

Rejestr TL pracuje jako dzielnik właściwy, natomiast rejestr TH służy jako bufor, którego zawartość jest przepisywana do rejestru TL gdy nastąpi jego przepełnienie. Operacja ta umożliwia otrzymanie dzielnika o zmiennym stopniu podziału. Należy przy tym pamiętać, że liczniki liczą w przód i dlatego do bufora trzeba wpisywać uzupełnienie do FFH liczby podziałowej.

Licznik T1 pracujący w trybie 2 jest często stosowany do taktowania portu szeregowego mikrokontrolera.

4.1.4 Tryb 3

Tryb 3 jest różny dla liczników T0 i T1. Schematyczny układ połączeń licznik T0 w trybie 3 przedstawia rys. 6.



Rys. 6. Tryb 3 dla licznika T0 mk 80C51/52

Rejestr TL0 jest sterowany znacznikami TR0, GATE0, CT0 oraz jest połączony z przerzutnikiem TF0. Może więc być sterowany i testowany jak licznik 0 w trybie 0, 1 i 2. Natomiast rejestr TH0 jest połączony z wyjściem zegara wewnętrznego o częstotliwości $f_{osc}/12$ poprzez bramkę sterowaną bitem TR1. Wyjście rejestru jest połączony z przerzutnikiem TF1.

Wprowadzenie trybu 3 dla licznika T1 powoduje jego zatrzymanie. Jeżeli licznik T0 pracuje w trybie 3, to licznik T1 może pracować w pozostałych trybach z tym, że nie może być sterowany znacznikiem TR1, a jego przepełnienie nie zmienia stanu przerzutnika TF1.

4.2 Przygotowanie licznika do pracy

Przed uruchomieniem licznika należy ustawić jego tryb pracy, dołączyć do zewnętrznego lub wewnętrznego źródła zliczanych impulsów, jeżeli potrzeba to uaktywnić przerwanie od przepełnienia licznika, wyzerować rejestry licznika i otworzyć bramkę.

5. Licznik T2

Układ posiada dwa rejestry TH2 i TL2 stanowiące 16-bitowy licznik (odpowiednio adresy CDH i CCH) oraz dwa rejestry zatraskujące RCAP2H i RCAP2L (CBH i CAH).

Rejestr T2CON mieści się pod adresem C8H w SFR i zawiera znaczniki:

T2CON	TF2	EXEF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2	adres C8H
-------	-----	-------	------	------	-------	-----	------	--------	-----------

TF2 - znacznik przepełnienia licznika 2. Ustawiany na 1 gdy nastąpi przepełnienie licznika. Nie dotyczy to sytuacji gdy $RCLK = 1$, lub $CLK = 1$. Znacznik ten musi zarezerwowany programowo.

EXF2 - znacznik ustawiony na 1 opadającym zboczem na wejściu T2EX(P1.1) gdy znacznik EXEN2 = 1. Jest to informacja, że nastąpiło autoładowanie licznika lub przepisanie stanu licznika do bufora pod wpływem zewnętrznego sygnału. Przy uaktywnionym przerwaniu od licznika 2 następuje zgłoszenie przerwania. Znacznik ten musi być zerowany programowo.

RCLK - znacznik przełączania generatora taktującego odbiornik portu szeregowego pracującego w trybie 1 lub 3. Gdy $RCLK = 1$ odbiornik portu szeregowego jest

taktowany sygnałem wyjściowym licznika2, a gdy $RCLK = 0$ - sygnałem wyjściowym licznika 1.

TCLK – znacznik przełączania generatora taktującego nadajnik portu szeregowego pracującego w trybie 1 lub 3. Zasada przełączania jest taka jak dla znacznika RCLK.

EXEN2-znacznik bramkujący zewnętrzne wejście sterujące T2EX (P1.1). Gdy $EXEN2 = 1$ licznik nie pracuje jako generator dla portu szeregowego, to opadające zbocze na wejściu T2EX, w zależności od trybu pracy licznika, może powodować autoładowanie lub przepisanie zawartości licznika do bufora.

TR2 -znacznik uruchamiania licznika. Gdy $TR2 = 1$ to licznik pracuje.

C/T2-znacznik przełączania wejścia licznika. Gdy $C/T2 = 0$, to licznik pracuje jako czasomierz z sygnałem odniesienia o częstotliwości $f_{osc}/12$ otrzymywanym z zegara wewnętrznego. Gdy $C/T2 = 1$ do wejścia licznika jest doprowadzony sygnał z wejścia T2 (P1.0).

CP/RL2 -znacznik przełączania trybu pracy. Gdy $CP/RL2 = 1$ – praca z zatrzaskiem. Do bufora jest przepisywana zawartość licznika gdy stan wejścia T2EX zmienia się z wysokiego na niski i znacznik $EXEN2 = 1$. Gdy $CP/RL2 = 0$ licznik pracuje z autoładowaniem, które odbywa się albo gdy następuje przeładowanie licznika, albo pod wpływem opadającego zbocza na wejściu T2EX (przy $EXEN2 = 1$). Znacznik ten jest ignorowany gdy licznik pracuje jako generator taktujący port szeregowy ($RCLK = 1$ lub $i TCLK = 1$).

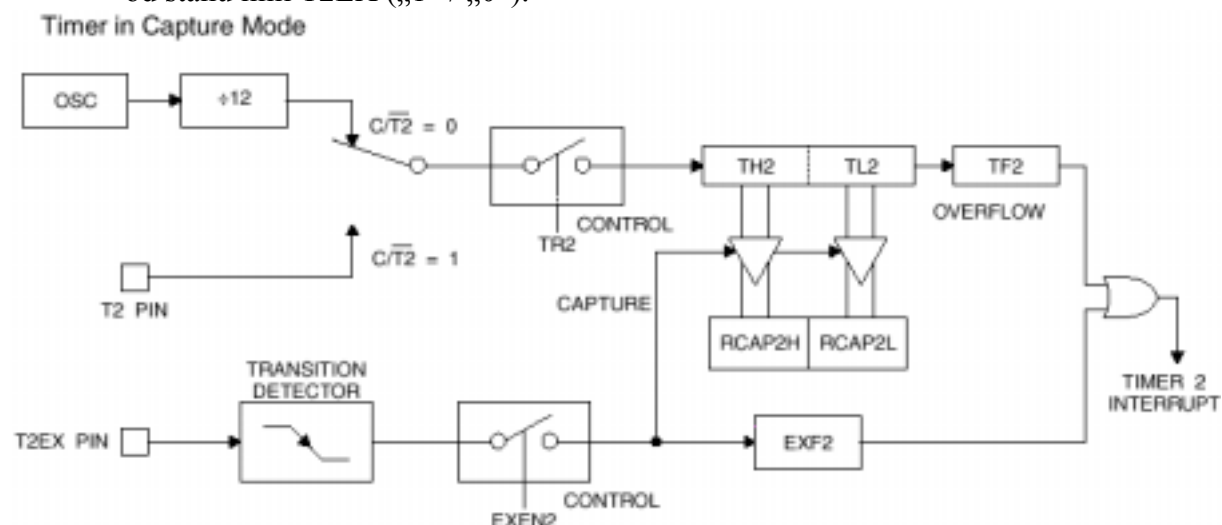
RCLK + TCLK	CP/RL2	TP2	TRYB PRACY
0	0	1	Licznik 16-to bitowy z autoładowaniem
0	1	1	Licznik 16-to bitowy z zatrzaskiem
1	X	1	Generator taktujący port szeregowy
X	X	0	Zatrzymanie licznika

Rejestr T2MOD mieści się pod adresem C9H w SFR i zawiera znaczniki:

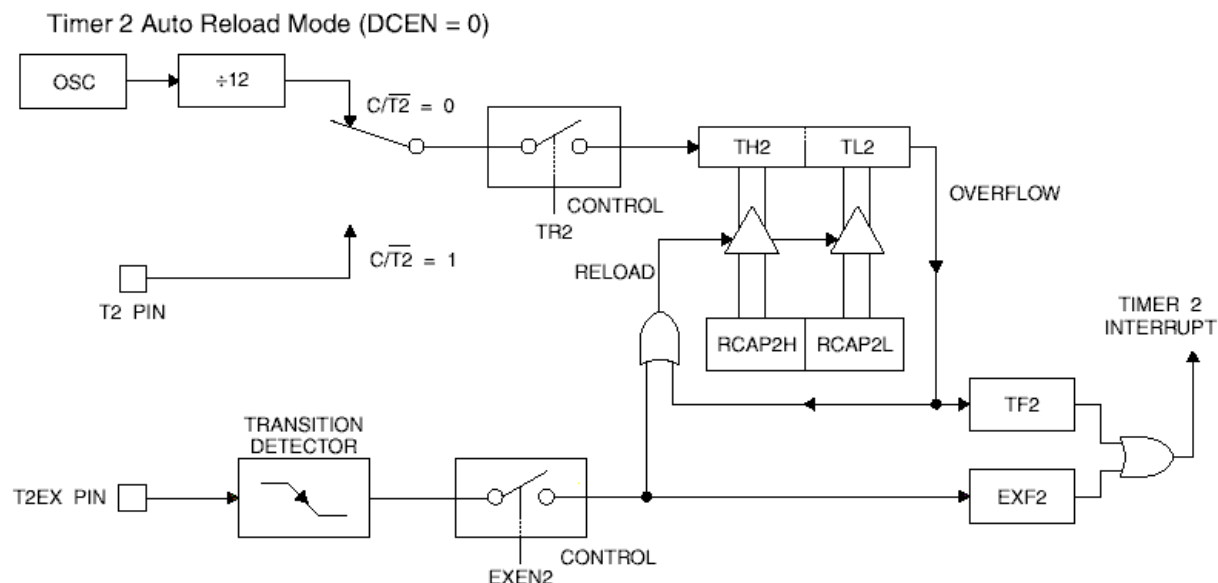
T2MOD - - - - - - T2OE DCEN adres C9H

T2OE - znacznik uaktywniający wyjście licznika T2.

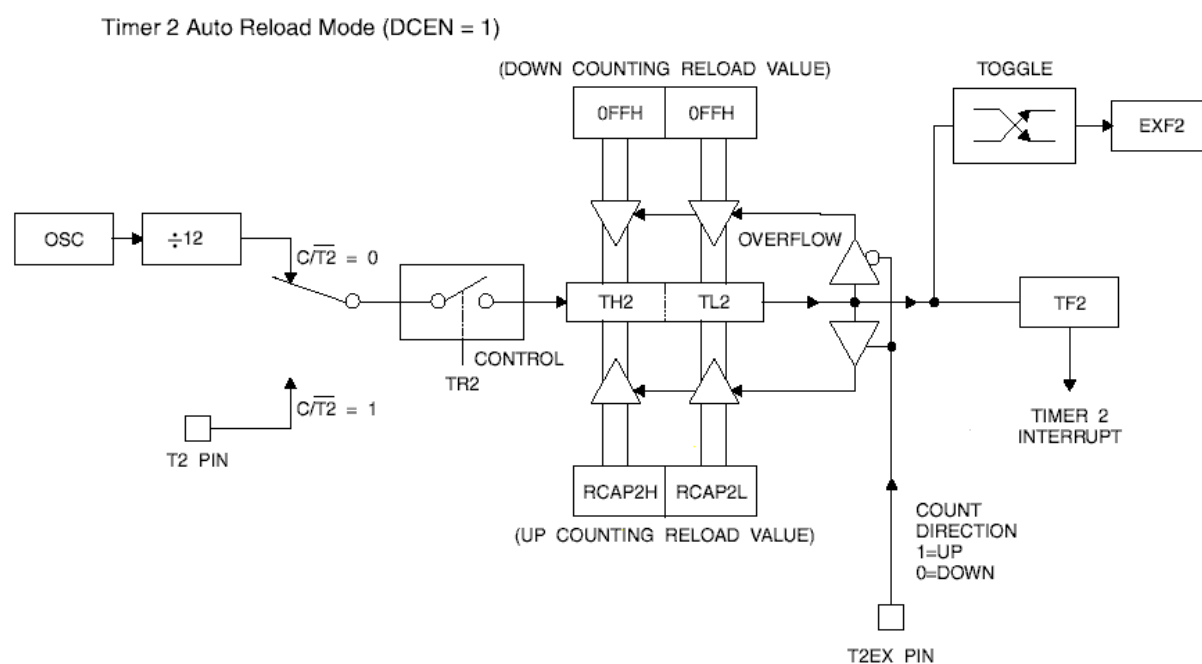
DCEN – jeżeli jest ustawiony umożliwia na zmianę kierunku zliczania góra/dół w zależności od stanu linii T2EX („1” / „0”).



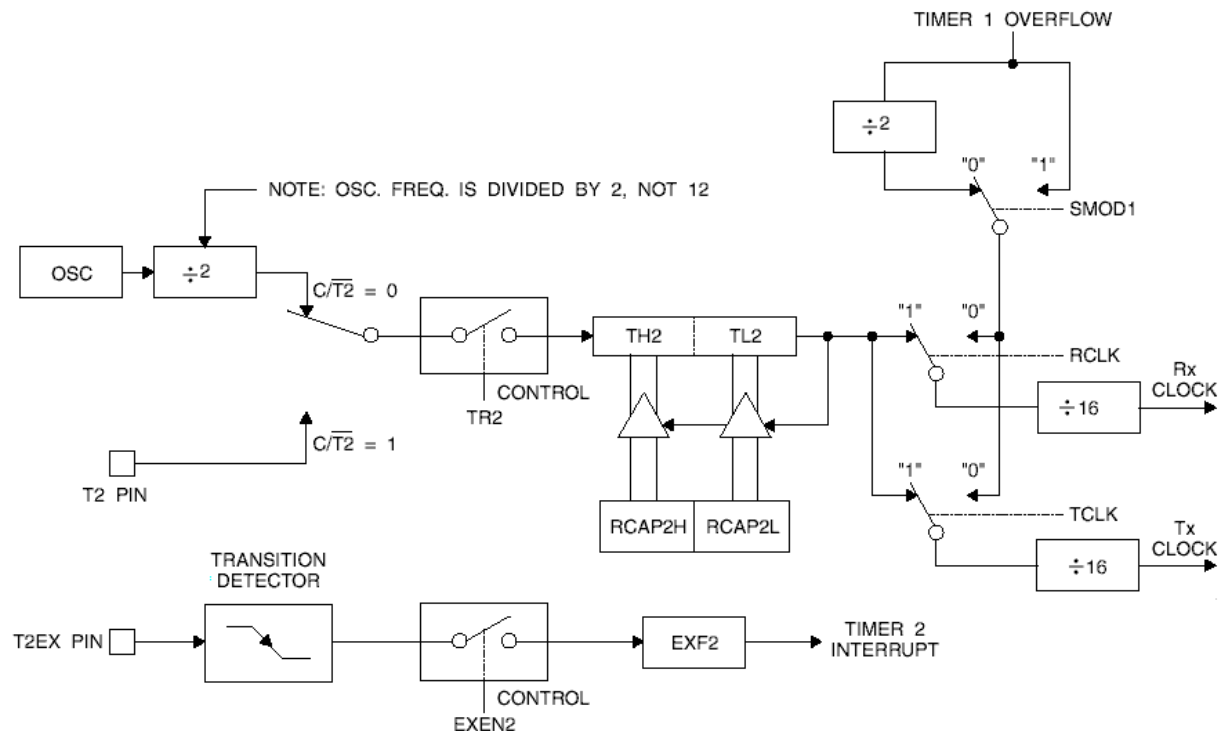
Rys. 7. Licznik T2 z autoładowaniem mk 80C51/52



Rys. 8. Licznik T2 z zatrzaskiem dla DCEN=0 mk 80C51/52

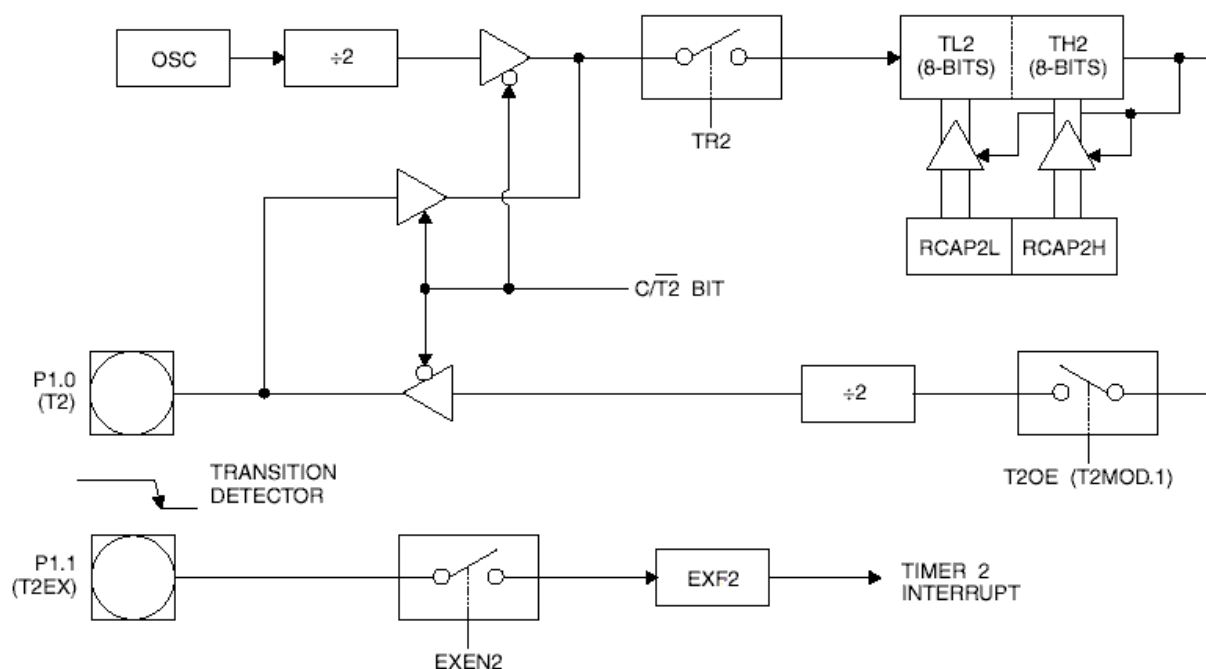


Rys. 9. Licznik T2 z zatrzaskiem dla DCEN=1 mk 80C51/52



Rys. 10. Licznik T2 jako generator sygnału taktującego UART mk 80C51/52

5. Timer 2 in Clock-out Mode



Rys. 10. Licznik T2 jako generator sygnału wyjściowego mk 80C51/52

6. Port szeregowy

Port szeregowy jest przeznaczony do komunikacji z urządzeniami zewnętrznymi. Dane są przesyłane poczynając od najmniej znaczącego bitu, synchronicznie lub asynchronicznie.

W trybie synchronicznym port może tylko nadawać lub przyjmować dane, w trybie asynchronicznym może równocześnie przyjmować i nadawać dane (full duplex).

Dostęp do portu odbywa się przez dwa rejestry SBUF mieszczące się pod adresem 99H w przestrzeni SFR. Zatem odczyt z rejestru SBUF może dawać inną wartość, niż tam wpisano.

Port szeregowy może pracować w czterech trybach. Tryby różnią się między sobą liczbą bitów w przesyłanych danych i źródłem sygnałów taktujących odbiornik i nadajnik. Tryb pracy portu oraz inne aspekty jego pracy są wybierane przez odpowiednie ustawienie znaczników w rejestrze SCON mieszczącego się pod adresem 98H w przestrzeni SFR. Znaczniki tego rejestru mogą być ustawiane pojedynczo.

Rejestr SCON zawiera następujące znaczniki:

SCON

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
-----	-----	-----	-----	-----	-----	----	----

 adres 98H

Znaczenie poszczególnych znaczników jest następujące:

SM0, SM1 - wybór trybu pracy

SM0	SM1	tryb	rodzaj transmisji	częst. taktująca
0	0	0	rejestr przesuwany (8 bitów)	$f_{osc}/12$
0	1	1	UART (8 bitów)	zmienna
1	0	2	UART (9 bitów)	$f_{osc}/64$ lub $f_{osc}/32$
1	1	3	UART (9 bitów)	zmienna

SM2 - maskowanie odbioru znaku

tryb 0	powinno być SM2 = 0
tryb 1	przy SM2 = 1 znacznik przerwania RI nie jest uaktywniany przy braku bitu stopu w odbieranym słowie
tryb 2 i 3	przy SM2 = 1 znacznik przerwania RI nie jest uaktywniany jeżeli 9-ty bit odbieranego słowa jest równy 0; w tych trybach znacznik ten jest wykorzystywany przy komunikacji wieloprocessorowej

REN - znacznik uaktywnienia odbiornika

0	odbiór zablokowany
1	odbiór dozwolony

TB8 - znacznik używany w transmisji dziewięciobitowej

tryb 0 i 1	nie używany
tryb 2 i 3	dziewiąty bit w nadawanym słowie

RB8 - znacznik używany w transmisji dziewięciobitowej

tryb 0	nie używany
tryb 1	gdy SM2, do RB8 jest wpisywany bit stopu odbieranego słowa
tryb 2 i 3	do RB8 jest wpisywany dziewiąty bit odbieranego słowa

TI - znacznik przerwania nadajnika. Ustawiany w stan 1 przez mikrokontroler tylnym zboczem 9-mego bitu wysyłanego słowa w trybie 0 lub przednim zboczem bitu stopu w pozostałych trybach. Może być zerowany tylko programowo.

RI - znacznik przerwania odbiornika. Ustawiany w stan 1 przez mikrokontroler tylnym zboczem 9-mego bitu przyjmowanego słowa w trybie 0 lub w połowie bitu stopu w pozostałych trybach z wyjątkiem gdy w trybach 2 i 3 bit SM2 = 1, a dziewiąty bit odbieranego słowa jest równy 0

6.1 Tryby pracy portu szeregowego

6.1.1 Tryb 0

W trybie 0 nadawanie i odbieranie słowa, w układzie rejestru przesuwanego, odbywa się przez końcówkę P3.0, a przez końcówkę P3.1 jest wysyłany sygnał taktujący, rysunek 7-1. Przesyłana dana jest 8-bitowa. Częstotliwość taktująca jest stała i wynosi $f_{osc}/12$.

Proces wysyłania danych rozpoczyna się w momencie wpisania do rejestru SBUF wysyłanej danej. Rejestr ten jest rejestrem przesuwalnym, jego zawartość jest przesuwana w prawo o jeden bit w fazie S6P2 każdego cyklu maszynowego. Na wolne miejsca po przesuniętych bitach wpisywane są zera.

Odbiór danych może odbywać się pod warunkiem ustawienia REN w stan 1 i wyzerowaniu znacznika RI. Stan wejścia RxD jest testowany w kolejnych cyklach maszynowych, a wynik testowania jest wpisywany do rejestru przesuwanego i przesuwany w lewo. Po skompletowaniu danej zawartość rejestru przesuwanego przepisywana jest do rejestru SBUF i następuje ustawienie znacznika RI w stan 1, co jest sygnałem zgłoszenia przerwania. Przyjęcie tego przerwania (gdy nie jest ono zamaskowane) nie powoduje automatycznego wyzerowania znacznika RI, a zatem jego stan można później badać programowo.

6.1.2 Tryby 1,2,3

W trybach 1, 2 i 3 port szeregowy pełni funkcję niezależnego nadajnika (linia P3.1 = TxD) i odbiornika (linia P3.0 = RxD) asynchronicznej transmisji szeregowej. Tryby różnią się między sobą długością przesyłanych słów oraz sposobem określenia szybkości transmisji.

Słowo przesyłane składa się z bitu startu (zawsze 0), ośmiu bitów danych w trybie 1 albo dziewięciu w trybie 2 i 3 oraz bitu stopu (zawsze 1). Słowo danych do wysłania jest pobierane z rejestru wyjściowego SBUF, odebrane słowo jest wpisywane do rejestru wejściowego SBUF. W trybach 2 i 3 ostatni D8 (MSB) bit danych jest pobierany do wysłania z pozycji TB8 (słowo SCON, adres 98H), a ten odbierany jest wpisywany na pozycję RB8 (słowo SCON).

Jeżeli w słowie SCON jest SM2 = 0, to wartość bitu D8 nie jest istotna dla działania układów transmisji szeregowej - można go wykorzystać programowo jako np. bit parzystości. Jeżeli natomiast jest SM2 = 1, to w porcie szeregowym jest maskowane odebranie znaku, w którym D8 = 0. Mechanizm ten jest wykorzystany w protokole transmisji dla systemu wieloprocessorowego.

Drugim elementem wpływającym na różnice między trybami 1, 2 i 3 jest źródło sygnału taktującego, określającego szybkość transmisji.

W trybie 2 układ transmisji jest taktowany wewnętrznym sygnałem zegarowym o częstotliwości $f_{osc}/2$, doprowadzonym bezpośrednio lub poprzez dzielnik przez 2 - w zależności od zawartości znacznika SMOD, tzn. PCON.7.

W trybach 1 i 3 sygnałem taktującym układ transmisji jest sygnał przepełnienia licznika T1 z układu czasowo-licznikowego. Jest on doprowadzany bezpośrednio lub poprzez dzielnik przez 2 - zgodnie z zawartością znacznika SMOD.

We wszystkich trzech trybach właściwy sygnał zegarowy, określający rytm pracy nadajnika (TxD) i odbiornika (RxD) powstaje po podzieleniu przez 16 odpowiedniego, opisanego

powyżej, sygnału taktującego. Zatem dla trybu 2 częstotliwość taktująca wynosi $f_{osc}/32$ lub $f_{osc}/64$, dla trybów 1 i 3 zależy od nastawy i trybu pracy licznika T1.

Wysłanie znaku jest inicjowane programowo, przez wykonanie dowolnego rozkazu zapisującego dane do rejestru SBUF. Po zakończeniu transmisji ustawiany jest znacznik TI rejestru SCON, adres 98H.

Odbieranie znaku jest inicjowane sprzętowo, po wykryciu zmiany 1 na 0 na linii P3.0 (Rx/D). Odbywa się to tylko wtedy, kiedy jest ustawiony znacznik REN w rejestrze SCON.

Aby odebrany znak znalazł się w rejestrze SBUF (i ew. dziewiąty bit w znaczniku RB8 - SCON.2 w trybach 2 i 3) muszą zachodzić dwa warunki:

1. RI (SCON.0) = 0
2. SM2 (SCON.5) = 0 lub
SM2 = 1 i odebrany bit D8 = 1

Jeśli warunki te są spełnione, ustawiany jest znacznik RI; jeśli zaś jeden z powyższych warunków nie zachodzi, to odebrany znak jest tracony.

W trybie 1 i 3, przy użyciu licznika w trybie 2 (8-bitowy licznik z autoładowaniem), prędkość transmisji jest określona zależnością:

$$f_{tr} = \frac{2^{SMOD}}{32} * \frac{f_{osc}}{12 * [256 - (TH1)]}$$

gdzie (TH1) - liczba wpisana do rejestru TH1 (adres 8DH).

Licznik 1 może również taktować port szeregowy pracując w pozostałych trybach, z sygnałem zewnętrznym lub wewnętrznym. W takim przypadku do ponownego załadowania licznika należy wykorzystać przerwanie wywołane przepełnieniem licznika a czas obsługi przerwania musi być uwzględniony przy obliczaniu częstotliwości przepełnienia licznika.

Korzystając z licznika T2 prędkość wyraża się następującym wzorem:

$$f_{tr} = \frac{f_{osc}}{32 * [65536 - (RCAP2H, RCAP2L)]}$$

gdzie: (RCAP2H, RCAP2L) - liczba wpisana do 16-bitowego rejestru licznika T2.

6.2 Przygotowanie portu szeregowego do pracy

Uruchomienie portu szeregowego, który ma pracować ze standardową prędkością bodową, wymaga ustawienia trybu pracy licznika - licznik z autoładowaniem, zaprogramowania licznika na określony stopień podziału, ustawienia trybu pracy portu szeregowego, a przy odbiorze danych należy ustawić znacznik REN. Po każdorazowym nadaniu i odbiorze bajtu danych należy wyzerować odpowiedni znacznik TI lub RI. Należy również podjąć decyzję, czy kontrola końca transmisji pojedynczego bajtu będzie się odbywać poprzez przerwanie, czy poprzez programowe testowanie znacznika TI przy nadawaniu lub RI przy odbiorze.

7. Przerwania

W mikrokontrolerze 8051 jest pięć źródeł przerwania, a w 8052 - sześć. Każde źródło przerwania może być odblokowane lub zablokowane przez ustawienie w stan 1 lub 0 odpowiedniego znacznika w rejestrze IE. Oto zawartość tego rejestru:

IE	EA	X	ET2	ES	ET1	EX1	ET0	EX0	adres 0A8H
----	----	---	-----	----	-----	-----	-----	-----	------------

Znaczenie poszczególnych bitów jest następujące:

EA - blokowanie wszystkich przerwań.

EA = 0 : żadne przerwanie nie może być przyjęte

EA = 1 : każde przerwanie może być przyjęte pod warunkiem, że odpowiadający mu znacznik jest w stanie 1

X - zarezerwowany

ETn - blokowanie (ETn = 0) lub odblokowanie (ETn = 1) przerwania od licznika Tn (n=0,1,2)

ES - blokowanie (ES = 0) lub odblokowanie (ES = 1) przerwania od portu szeregowego

EXn - blokowanie (EXn = 0) lub odblokowanie (EXn = 1) przerwania zewnętrznego z wejścia INTn (n = 0,1)

Sposób reakcji na sygnał przerwania zewnętrznego jest ustawiany w rejestrze TCON:

TCON

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

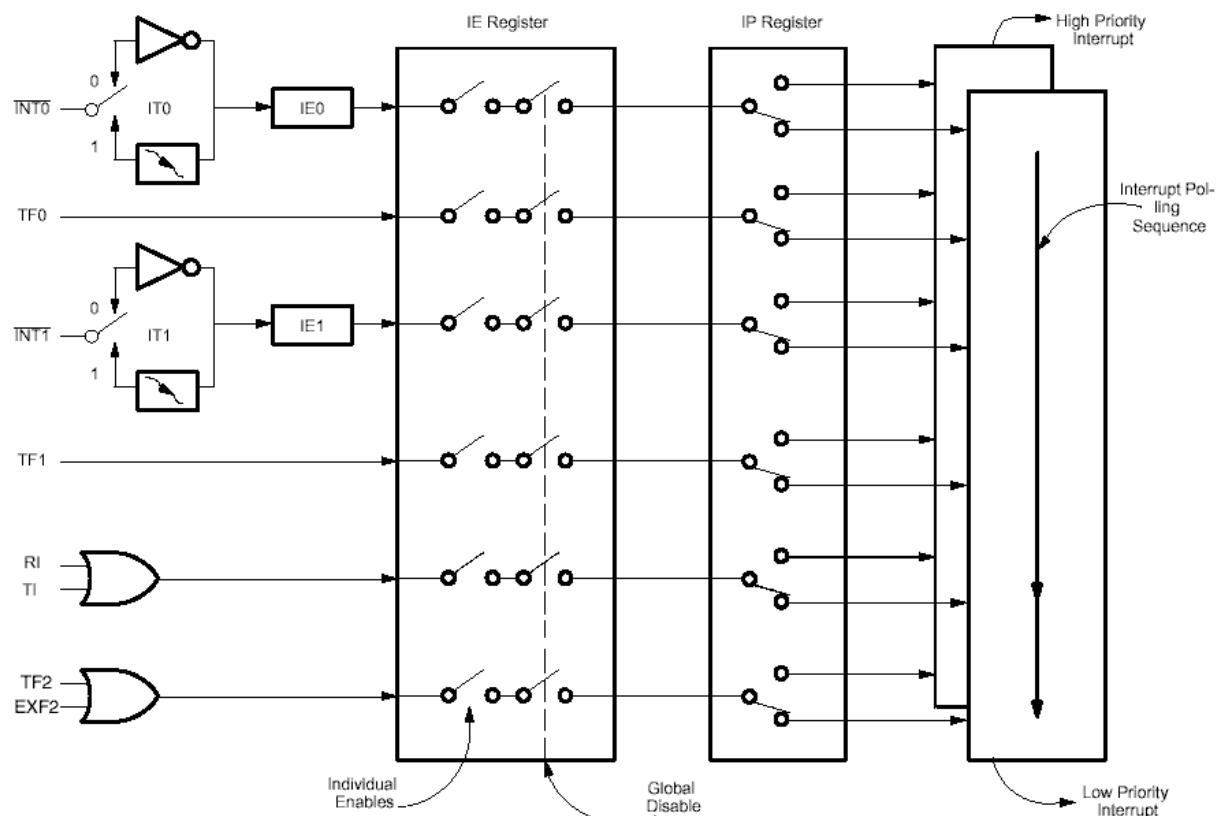
 adres 88H

Znaczenie bitów (n = 0,1):

IE_n - znacznik zgłoszenia przerwania na INT_n; ustawiany sprzętowo po wykryciu zgłoszenia zewnętrznego na wejściu INT_n; zerowany po przyjęciu przerwania,

IT_n - bit sterujący zgłoszeniem przerwania na INT_n; zmieniany programowo, określa sposób zgłaszania przerwania zewnętrznego na INT_n, tzn. poziomem niskim (IT_n = 0) lub opadającym zboczem (IT_n = 1) sygnału przerywającego.

Pozostałe bity są związane z pracą liczników (zobacz p. 4 *Liczniki* oraz opis poniżej).



Rys. 11. Schemat funkcjonalny systemu przerwań w mk 80C52

Po wykryciu odpowiedniego sygnału przerwania znaczniki IE0 i IE1 zmieniają swój stan na 1, co powoduje przejście procesora do obsługi programu przerwania, jeżeli EA=1.

Jeżeli przerwanie jest wywoływane zboczem, to znacznik IEn jest zerowany przez procesor w momencie przejścia do obsługi programu przerwania. Natomiast gdy przerwanie reaguje na poziom, to stan znacznika IEn zależy od stanu sygnału na wejściu. Dlatego nastąpi ponowne wywołanie przerwania gdy stan sygnału przerwania nie zmieni się na 1 przed końcem programu obsługi przerwania.

Przerwanie pochodzące od liczników T0 i T1 są generowane, gdy zostaną ustawione znaczniki TF0 lub TF1 w rejestrze TCON, co następuje w czasie przepelnienia liczników (z wyjątkiem pracy licznika 0 w trybie 3). Obydwa znaczniki są zerowane w momencie przejścia do obsługi przerwania.

Przerwanie pochodzące od portu szeregowego jest wywoływane przez ustawienie znacznika RI lub TI w rejestrze SCON (adres 98H). Układ przerwań nie rozróżnia źródła przerwania (odbiornik czy nadajnik) - musi to zrobić program obsługi przerwania - i dlatego znaczniki te nie są zerowane przez procesor przy przejściu do obsługi przerwania.

Wszystkie znaczniki wywołujące przerwania mogą być programowo ustawiane i zerowane. Programowe ustawienie tych bitów wywoła przerwanie, o ile będzie ono dozwolone.

Przy jednoczesnym nadejściu kilku sygnałów przerwań o kolejności obsługi przerwania decyduje priorytet przerwań przedstawiony w poniższej tabeli.

Lp.	źródło przerwania	priorytet	adres obsługi
1.	IE0	najwyższy	03H
2.	TF0		0BH
3.	IE1		13H
4.	TF1		1BH
5.	RI + TI		23H
6.	TF2 + EXF2 (tylko 8052)	najniższy	2BH

Jeżeli program obsługujący przerwanie ma być przerywany przez inne przerwanie, to w rejestrze IP priorytetu przerwań musi być ustawiony znacznik odpowiadający źródłu przerywającemu:

IP

X	X	PT2	PS	PT1	PX1	PT0	PX0
---	---	-----	----	-----	-----	-----	-----

 adres 0B8H

Znaczenie bitów jest następujące:

X - zarezerwowane

PTn - licznik n (n=0,1,2)

PS - port szeregowy

PXn - przerwanie zewnętrzne z wejścia INTn (n=0,1)

Ustawienie w rejestrze IP znacznika dla danego źródła przerwania powoduje, że przerwanie to osiąga wyższy priorytet od przerwań, dla których znaczniki mają stan 0. Natomiast wzajemna relacja pomiędzy źródłami przerwań, których znaczniki mają ten sam stan, nie ulega zmianie.

Mikrokontroler nie obsługuje sygnału przerwania, który nadszedł w momencie gdy przerwanie było zablokowane.

Obsługa przerwania rozpoczyna się od wygenerowania instrukcji LCALL. W tym samym czasie zerowany jest znacznik wywołujący przerwanie, chyba że tym znacznikiem jest znacznik portu szeregowego lub licznika T2, które nie są zerowane. Również nie są zerowane znaczniki przerwania zewnętrznego, gdy przerwanie ma reagować na poziom. Instrukcja LCALL powoduje schowanie zawartości licznika rozkazów PC na stos (ale nie jest chowane słowo statusu PSW) a do licznik rozkazów jest ładowany adres pierwszej instrukcji programu obsługi przerwania. Każde przerwanie ma przydzielony w pamięci programu adres. Adresy te zebrano w powyższej tabeli (por. p. 2.1 *Pamięć kodu programu*).

Wykonywanie programu obsługi kończy się wykonaniem instrukcji RETI. W efekcie do rejestru PC przepisywane są dwa bajty ze szczytu stosu oraz układ przerw zostaje ustawiony w stanie wyjściowym.

Zakończenie przerwania instrukcją RET również spowoduje kontynuację przerwanej programu, ale procesor nie otrzyma sygnału o końcu przerwania, co w konsekwencji spowoduje nieprzyjęcie następnego przerwania o priorytecie niższym od ostatnio obsługowanego.

8. Start procesora

Po włączeniu zasilania lub w przypadku zawieszenia się programu należy ustalić warunki początkowe, od których procesor rozpocznie pracę. Do tego celu przeznaczone jest wyprowadzenie zerujące RST. W celu wyzerowania mikrokontrolera należy do wyprowadzenia RST doprowadzić, na okres dwóch cykli maszynowych (przy pracującym oscylatorze), napięcie odpowiadające stanowi 1. W drugim cyklu maszynowym następuje wytworzenie wewnętrznego sygnału zerującego, który jest powtarzany co każdy cykl maszynowy aż do momentu, w którym wejście RST znajdzie się w stanie 0. Wewnętrzny sygnał zerowania przełącza wyprowadzenia ALE i PSEN jako wejścia i ustawia rejestry mikrokontrolera w sposób następujący:

Rejestr	Zawartość
PC	0000
ACC	00
B	00
PSW	00
SP	07
DPTR	0000
P0 - P3	0FFH
IP	XX000000B
IE	0X000000B
TMOD	00
TCON	00
T2CON	00
TH0	00
TL0	00
TH1	00
TL1	00
TH2	00
TL2	00
SCON	00

SBUF	nieokreślony
PCON	0XXX0000B

W czasie zerowania procesora zawartość wewnętrznej pamięci danych RAM nie zmienia swojej zawartości. Po włączeniu zasilania zawartość pamięci jest przypadkowa, ale po zerowaniu, gdy napięcie zasilania nie było wyłączane, pamięć zachowuje wpisane do niej dane.

9. Obniżanie poboru mocy

Dla procesorów wykonanych w technologii CHMOS standardowo istnieją dwa stany o zredukowanym poborze mocy: stan zatrzymania (Idle) i stan wyłączenia zasilania (Power Down). Natomiast dla procesorów wykonanych w technologii HMOS brak rozwiązań standardowych.

Wejściem zasilania awaryjnego w układach HMOS jest wejście zerujące RST. W czasie normalnej pracy pamięć RAM - jak i cały układ - jest zasilana z wejścia Ucc. Jeżeli RST = +5V to przy zaniku zasilania na Ucc wejście RST przejmuje funkcję wejścia zasilającego pamięć danych. Powrót do normalnej pracy po ponownym włączeniu zasilania za Ucc następuje przez zerowanie systemu. Po włączeniu zasilania stan wysoki na RST musi trwać jeszcze przez pewien czas, potrzebny do wyzerowania mikrosterownika. Procedura startu procesora powinna rozróżniać, czy procesor startuje od początku, czy po zaniku napięcia zasilania. Rozróżnienie może odbywać się np. poprzez testowanie odpowiedniego znacznika.

W procesorach CMOS wprowadzenie do stanu obniżonego poboru mocy odbywa się przez ustawienie odpowiedniego znacznika w rejestrze PCON:

PCON

SMOD	X	X	X	GF1	GF0	PD	IDL
------	---	---	---	-----	-----	----	-----

 adres 87H

Znaczenie bitów:

GF_n - znacznik ogólnego przeznaczenia (n=0,1)

PD - znacznik stanu wyłączenia

PD = 0 : praca normalna

PD = 1 : przejście do stanu wyłączenia

IDL - znacznik stanu zatrzymania

IDL = 0 : praca normalna

IDL = 1 : przejście do stanu zatrzymania

X - zarezerwowane

SMOD - znacznik portu szeregowego, zob. p. 5 *Port szeregowy*

W stanie zatrzymania i wyłączenia podtrzymanie zawartości wewnętrznej pamięci danych odbywa się poprzez wyprowadzenia Vcc (a nie, jak w technologii HMOS, przez wejście RST).

Instrukcja ustawienia znacznika IDL jest ostatnią instrukcją wykonaną przed przejściem procesora w **stan zatrzymania**. Stan zatrzymania procesora polega na zablokowaniu sygnałów zegarowych sterujących pracą jednostki centralnej CPU, natomiast pracuje układ przerwań, port szeregowy i liczniki. Pamięć RAM i wszystkie rejestry, w tym i licznik rozkazów, licznik

stosu, akumulator, zachowują swoją zawartość. Wyprowadzenia portów utrzymują stan jaki miały w momencie wejścia w stan zatrzymania. Sygnały ALE i PSEN są w stanie 1.

Istnieją dwie drogi wyjścia ze stanu zatrzymania. Pierwsza polega na wywołaniu nie zablokowanego przerwania, co powoduje zerowanie bitu IDL. Następuje obsługa przerwania, a pierwszą instrukcją po powrocie z przerwania (po instrukcji RETI) jest instrukcja występująca bezpośrednio po instrukcji wprowadzającej procesor w stan zatrzymania.

Znaczniki GF0 i GF1 mogą być użyte do identyfikacji, czy przerwania nastąpiło w czasie normalnej pracy, czy w stanie zatrzymania. Jednak w wykonaniu HMOS w rejestrze specjalnym PCON jest dostępny tylko znacznik SMOD.

Drugą drogą wyjścia ze stanu zatrzymania jest zerowanie procesora. Ponieważ oscylator cały czas pracuje, wystarczy by sygnał zerujący trwał przez dwa cykle maszynowe.

Stan wyłączenia procesora uzyskuje się przez ustawienie bitu PD. Jest to ostatnia wykonywana instrukcja przed wyłączeniem pracy procesora. W stanie wyłączenia jest blokowana praca oscylatora, co powoduje, że żaden z modułów wewnętrznych procesora nie może pracować. Zawartość pamięci RAM i rejestrów jest zachowana, a na liniach wyjściowych portów jest podana zawartość odpowiadających im rejestrów. Wyjścia ALE i PSEN są w stanie 0. Jedynym sposobem wyjścia ze stanu wyłączenia jest zerowanie procesora.

IV. Dokumentacja – opis instrukcji i dyrektyw asemblera mikrokontrolera 80C51/52

1. Instrukcje asemblera dla 80C51/52

MOV A,Rr	Mov register to accumulator	$A \leftarrow Rr$	1/1	Rr - rejestr R0 ÷ R7
MOV A,ad	Mov address mediately to acc.	$A \leftarrow \langle ad \rangle$	2/1	ad - bezpośredni adres 8-bitowy
MOV A,@Ri	Mov address immediately to acc.	$A \leftarrow \langle Ri \rangle$	1/1	Ri - rejestr R0 ÷ R1
MOV A,#n	Mov data immediately to acc.	$A \leftarrow n$	2/1	n - 8-bitowy argument bezpośredni
MOV Rr,A	Mov accumulator to register	$Rr \leftarrow A$	1/1	Rr - rejestr R0 ÷ R7
MOV Rr,ad	Mov address mediately to register	$Rr \leftarrow \langle ad \rangle$	2/2	Rr - rejestr R0 ÷ R7 ad - bezpośredni adres 8-bitowy
MOV Rr,#n	Mov immediately data to register	$Rr \leftarrow n$	2/1	Rr - rejestr R0 ÷ R7 n - 8-bitowy argument bezp.
MOV ad,A	Mov accumulator to address	$\langle ad \rangle \leftarrow A$	2/1	ad - bezpośredni adres 8-bitowy
MOV ad,Rr	Mov register to address	$\langle ad \rangle \leftarrow Rr$	2/2	ad - bezpośredni adres 8-bitowy Rr - rejestr R0 ÷ R7
MOV ad1,ad2	Mov adr2 to adr1	$\langle ad1 \rangle \leftarrow \langle ad2 \rangle$	3/2	ad1, ad2 - 8-bitowe adresy bezp.
MOV ad,@Ri	Mov mediately register to address	$\langle ad \rangle \leftarrow \langle Ri \rangle$	2/2	ad - bezpośredni adres 8-bitowy Ri - rejestr R0 ÷ R1
MOV ad,#n	Mov data to address	$\langle ad \rangle \leftarrow n$	3/2	ad - bezpośredni adres 8-bitowy n - 8-bitowy argument bezp.
MOV @Ri,A	Mov accumulator to mediately reg.	$\langle Ri \rangle \leftarrow A$	1/1	Ri - rejestr R0 ÷ R1
MOV @Ri,ad	Mov address to mediately register	$\langle Ri \rangle \leftarrow \langle ad \rangle$	2/2	Ri - rejestr R0 ÷ R1 ad - bezpośredni adres 8-bitowy
MOV @Ri,#n	Mov immediately data to med. reg.	$\langle Ri \rangle \leftarrow n$	2/1	Ri - rejestr R0 ÷ R1 n - 8-bitowy argument bezp.
MOV DPTR,#nn	Mov immediately data to DPTR	$DPTR \leftarrow nn$	3/2	nn - 16-bitowy argument bezpośredni
XCH A,Rr	Exchange acc. witch memory	$A \leftrightarrow Rr$	1/1	Rr - rejestr R0 ÷ R7
XCH A,ad	Exchange acc. witch memory	$A \leftrightarrow \langle ad \rangle$	2/1	ad - bezpośredni adres 8-bitowy
XCH A,@Ri	Exchange acc. witch memory	$A \leftrightarrow \langle Ri \rangle$	1/1	Ri - rejestr R0 ÷ R1
XCHD A,@Ri	Exchange nibble of acc. witch mem	$A_{3-0} \leftrightarrow \langle Ri \rangle_{3-0}$	1/1	Ri ₃₋₀ - młodsza część rejestru Ri
MOVX A,@Ri	Move from external mem. to acc.	$A \leftarrow \langle Ri \rangle$	1/2	Ri - rejestr R0 ÷ R1, zawiera adres 8-bitowy
MOVX @Ri,A	Nove from acc. to external memory	$\langle Ri \rangle \leftarrow A$	1/2	Ri - rejestr R0 ÷ R1, zawiera adres 8-bitowy
MOVX A,@DPTR	Move from external mem. to acc.	$A \leftarrow (DPTR)$	1/2	DPTR - zawiera adres 16-bitowy
MOVX @DPTR,A	Nove from acc. to external memory	$(DPTR) \leftarrow A$	1/2	DPTR - zawiera adres 16-bitowy
MOVC A,@A+DPTR	Move from program mem. to acc	$A \leftarrow (A + DPTR)$	1/2	DPTR - 16-bitowy wskaźnik danych
MOVC A,@A+PC	Move from program mem. to acc	$A \leftarrow (A + PC)$	1/2	PC - 16-bitowy licznik rozkazów
ADD A,Rr	Add to acc. register	$A \leftarrow A + Rr$	C,AC,OV,P	Rr - rejestr R0 ÷ R7
ADD A,ad	Add to acc. addr. contenst	$A \leftarrow A + (ad)$	C,AC,OV,P	ad - bezpośredni adres 8-bitowy
ADD A,@Ri	Add toacc. mediately addr. contest	$A \leftarrow A + (Ri)$	C,AC,OV,P	Ri - rejestr R0 ÷ R1
ADD A,#n	Add to acc. immediately data	$A \leftarrow A + n$	C,AC,OV,P	n - 8-bitowy argument bezp.

ADDC A,Rr	Add to acc. reg. with carry	$A \leftarrow A + Rr + CY$	C,AC,OV,P	Rr - rejestr R0 ÷ R7
ADDC A,ad	Add to acc. mediately addr. contest	$A \leftarrow A + \langle ad \rangle + CY$	C,AC,OV,P	ad - bezpośredni adres 8 - bitowy
ADDC A,@Ri	Add to acc. immed. addr. contest	$A \leftarrow A + \langle Ri \rangle + CY$	C,AC,OV,P	Ri - rejestr R0 ÷ R1
ADDC A,#n	Add to acc. immediately data	$A \leftarrow A + n + CY$	C,AC,OV,P	n - 8 - bitowy argument bezp.
SUBB A,Rr	Subb. from acc. with borrow reg.	$A \leftarrow A - Rr - CY$	C,AC,OV,P	Rr - rejestr R0 ÷ R7
SUBB A,ad	Subb. from acc. with borrow addr. contest	$A \leftarrow A - \langle ad \rangle - CY$	C,AC,OV,P	ad - bezpośredni adres 8 - bitowy
SUBB A,@Ri	Subb. from acc. with borrow mediat. addr. contest	$A \leftarrow A - \langle Ri \rangle - CY$	C,AC,OV,P	Ri - rejestr R0 ÷ R1
SUBB A,#n	Subb. from acc. with borrow immediately data	$A \leftarrow A - n - CY$	C,AC,OV,P	n - 8 - bitowy argument bezp.
INC A	Increment acc.	$A \leftarrow A + 1$	P	
INC Rr	Increment register	$Rr \leftarrow Rr + 1$	-	Rr - rejestr R0 ÷ R7
INC ad	Inc. addr. contest	$\langle ad \rangle \leftarrow \langle ad \rangle + 1$	-	ad - bezpośredni adres 8 - bitowy
INC @Ri	Inc. mediately address contest	$\langle Ri \rangle \leftarrow \langle Ri \rangle + 1$	-	Ri - rejestr R0 ÷ R1
INC DPTR	Increment DPTR	$DPTR \leftarrow DPTR + 1$	-	
DEC A	Decrement acc.	$A \leftarrow A - 1$	P	
DEC Rr	Decrement reg.	$Rr \leftarrow Rr - 1$	-	Rr - rejestr R0 ÷ R7
DEC ad	Dec. addr. contest	$\langle ad \rangle \leftarrow \langle ad \rangle - 1$	-	ad - bezpośredni adres 8 - bitowy
DEC @Ri	Dec. mediately address contest	$\langle Ri \rangle \leftarrow \langle Ri \rangle - 1$	-	Ri - rejestr R0 ÷ R1
MUL AB	Multiply A per B	$B.A \leftarrow A * B$	C=0,OV,P	rej B - 8 starszych bitów wyniku rej A - 8 młodszych bitów wyniku
DIV AB	Division A per B	$A \leftarrow [A/B]$ cz.calk. $B \leftarrow [A/B]$ reszta	C=0,OV,P	rej B - 8 starszych bitów wyniku rej A - 8 młodszych bitów wyniku
DA A	Decimal adjust	korekcja dziesiętna	C,AC,P	
ANL A,Rr	Logical AND A with register	$A \leftarrow A \text{ and } Rr$	P	Rr - rejestr R0 ÷ R7
ANL A,ad	Logical AND A and addr. contest	$A \leftarrow A \text{ and } \langle ad \rangle$	P	ad - bezpośredni adres 8 - bitowy
ANL A,@Ri	Logical AND A with mediately address contest	$A \leftarrow A \text{ and } \langle Ri \rangle$	P	Ri - rejestr R0 ÷ R1
ANL A,#n	Logical AND A with immediately data	$A \leftarrow A \text{ and } n$	P	n - 8 - bitowy argument bezp.
ANL ad,A	Logical AND addr. contents and A	$\langle ad \rangle \leftarrow \langle ad \rangle \text{ and } A$	-	ad - bezpośredni adres 8 - bitowy
ANL ad,#n	Logical AND addr and immediately data	$\langle ad \rangle \leftarrow \langle ad \rangle \text{ and } n$	-	n - 8 - bitowy argument bezp.
ORL A,Rr	Logical OR A with register	$A \leftarrow A \text{ or } Rr$	P	Rr - rejestr R0 ÷ R7
ORL A,ad	Logical OR A with addr. contest	$A \leftarrow A \text{ or } \langle ad \rangle$	P	ad - bezpośredni adres 8 - bitowy
ORL A,@Ri	Logical OR A with mediately address contest	$A \leftarrow A \text{ or } \langle Ri \rangle$	P	Ri - rejestr R0 ÷ R1
ORL A,#n	Logical OR A with immediately data	$A \leftarrow A \text{ or } n$	P	n - 8 - bitowy argument bezp.
ORL ad,A	Logical OR addr. contest with A	$\langle ad \rangle \leftarrow \langle ad \rangle \text{ or } A$	-	ad - bezpośredni adres 8 - bitowy

ORL <i>ad,#n</i>	Logical OR addr. with immediately data	$\langle ad \rangle \leftarrow \langle ad \rangle \text{ or } n$	-	<i>ad</i> - bezpośredni adres 8-bitowy <i>n</i> - 8-bitowy argument bezp.
XRL <i>A,Rr</i>	Logical XOR <i>A</i> with <i>Rr</i>	$A \leftarrow A \text{ xor } Rr$	P	<i>Rr</i> - rejestr R0 ÷ R7
XRL <i>A,ad</i>	Logical XOR <i>A</i> with addr. contest	$A \leftarrow A \text{ xor } \langle ad \rangle$	P	<i>ad</i> - bezpośredni adres 8-bitowy
XRL <i>A,@Ri</i>	Logical XOR <i>A</i> with mediately address contest	$A \leftarrow A \text{ xor } \langle Ri \rangle$	P	<i>Ri</i> - rejestr R0 ÷ R1
XRL <i>A,#n</i>	Logical XOR <i>A</i> with immediately data	$A \leftarrow A \text{ xor } n$	P	<i>n</i> - 8-bitowy argument bezp.
XRL <i>ad,A</i>	Logical XOR addr. conest with <i>A</i>	$\langle ad \rangle \leftarrow \langle ad \rangle \text{ xor } A$	-	<i>ad</i> - bezpośredni adres 8-bitowy
XRL <i>ad,#n</i>	Logical XOR addr. with immediately data	$\langle ad \rangle \leftarrow \langle ad \rangle \text{ xor } n$	-	<i>ad</i> - bezpośredni adres 8-bitowy <i>n</i> - 8-bitowy argument bezp.
CLR <i>A</i>	Clear acc.	$A \leftarrow 0$	P	
CPL <i>A</i>	Complement acc.	$A \leftarrow \text{not}(A)$	-	
SWAP <i>A</i>	Swap nibbles within acc.	$A_{3:0} \leftarrow A_{7:4}$	-	
RL <i>A</i>	Rotate left acc.	$\uparrow 7.. \leftarrow ..0 \downarrow$	-	
RLC <i>A</i>	Rotate left acc. through carry	$\uparrow CY \leftarrow 7 \leftarrow 0 \downarrow$	C,P	
RR <i>A</i>	Rotate right acc.	$\downarrow 7.. \rightarrow ..0 \uparrow$	-	
RRC <i>A</i>	Rotate right acc. through carry	$\downarrow CY \rightarrow 7 \rightarrow 0 \uparrow$	C,P	

CLR <i>C</i>	Clear carry bit	$CY \leftarrow 0$	C	
CLR <i>bit</i>	Clear bit	$\langle bit \rangle \leftarrow 0$	-	<i>bit</i> -8-bitowy adres bitu w pamięci
SETB <i>C</i>	Set carry bit	$CY \leftarrow 1$	C	
SETB <i>bit</i>	Set bit	$\langle bit \rangle \leftarrow 1$	-	<i>bit</i> -8-bitowy adres bitu w pamięci
CPL <i>C</i>	Complement carry bit	$CY \leftarrow \text{not}(CY)$	C	
CPL <i>bit</i>	Complement bit	$\langle bit \rangle \leftarrow \text{not}(\langle bit \rangle)$	-	<i>bit</i> -8-bitowy adres bitu w pamięci
ANL <i>C,bit</i>	Logical AND <i>CY</i> with bit	$CY \leftarrow CY \text{ and } \langle bit \rangle$	C	<i>bit</i> -8-bitowy adres bitu w pamięci
ANL <i>C,/bit</i>	Logic AND <i>CY</i> with complement bit	$CY \leftarrow CY \text{ and not}(\langle bit \rangle)$	C	<i>bit</i> -8-bitowy adres bitu w pamięci
ORL <i>C,bit</i>	Logic OR <i>CY</i> witch bit	$CY \leftarrow CY \text{ or } \text{Bit}$	C	<i>bit</i> -8-bitowy adres bitu w pamięci
ORL <i>C,/bit</i>	Logic OR <i>CY</i> witch comp.bit	$CY \leftarrow \text{or not}(\langle bit \rangle)$	C	<i>bit</i> -8-bitowy adres bitu w pamięci
MOV <i>C,bit</i>	Move bit to <i>CY</i>	$CY \leftarrow \langle bit \rangle$	C	<i>bit</i> -8-bitowy adres bitu w pamięci
MOV <i>bit,C</i>	Mov <i>CY</i> to bit	$\langle bit \rangle \leftarrow CY$	-	<i>bit</i> -8-bitowy adres bitu w pamięci
AJMP <i>adr11</i>	Unconditional jump on page	$PC_{10:0} \leftarrow \text{adr11}$	-	<i>adr11</i> - adres 11-bitowy
LJMP <i>adr16</i>	Uncoditional jump	$PC \leftarrow \text{adr16}$	-	<i>adr16</i> -adres 16-bitowy
SJMP <i>d</i>	Jump relative	$PC \leftarrow PC + d$	-	<i>d</i> - 8-bitowe przesunięcie
JMP@A+DPTR	Jump indirect	$PC \leftarrow PC + DPTR$	-	
JC <i>d</i>	Jump if carry is set	$\text{gdy } CY=1 \text{ to } PC \leftarrow PC + d$	-	<i>d</i> - 8-bitowe przesunięcie
JNC <i>d</i>	Jump if carry is not set	$\text{gdy } CY=0 \text{ to } PC \leftarrow PC + d$	-	<i>d</i> - 8-bitowe przesunięcie
JZ <i>d</i>	Jump if acc. zero	$\text{gdy } A=0 \text{ to } PC \leftarrow PC + d$	-	<i>d</i> - 8-bitowe przesunięcie
JNZ <i>d</i>	Jump if acc. not zero	$\text{gdy } A \neq 0 \text{ to } PC \leftarrow PC + d$	-	<i>d</i> - 8-bitowe przesunięcie
JB <i>bit,d</i>	Jump if bit is set	$\text{gdy } \langle bit \rangle = 1 \text{ to } PC \leftarrow PC + d$	-	<i>bit</i> -8-bitowy adres bitu w pamięci <i>d</i> - 8-bitowe przesunięcie

JNB <i>bit,d</i>	Jump if bit is not set	gdy $\langle bit \rangle = 0$ to $PC \leftarrow PC + d$	-	<i>bi</i> - 8-bitowy adres bitu w pamięci <i>d</i> - 8-bitowe przesunięcie
JBC <i>bit,d</i>	Jump if bit is set and clear bit	gdy $\langle bit \rangle = 1$ to $PC \leftarrow PC + d$ $\langle bit \rangle \leftarrow 0$	-	<i>bit</i> - 8-bitowy adres bitu w pamięci <i>d</i> - 8-bitowe przesunięcie
CJNE <i>A,ad,d</i>	Compare acc. and jump if not equal with addr. contest	gdy $A \neq \langle ad \rangle$ to $PC \leftarrow PC + d$	C	<i>ad</i> - bezpośredni adres 8-bitowy <i>d</i> - 8-bitowe przesunięcie
CJNE <i>A,#n,d</i>	Compare acc. and jump if not equal with immed. data	gdy $A \neq n$ to $PC \leftarrow PC + d$	C	<i>n</i> - 8-bitowy argument bezp. <i>d</i> - 8-bitowe przesunięcie
CJNE <i>Rr,#n,d</i>	Compare reg. and jump if not equal with immed. data	gdy $Rr \neq n$ to $PC \leftarrow PC + d$	C	<i>Rr</i> - rejestr R0 ÷ R7 <i>n</i> - 8-bitowy argument bezp. <i>d</i> - 8-bitowe przesunięcie
CJNE <i>@Ri,#n,d</i>	Compare mediately reg. contest and jump if not equal with immed. data	gdy $\langle Ri \rangle \neq n$ to $PC \leftarrow PC + d$	C	<i>Ri</i> - rejestr R0 ÷ R1 <i>n</i> - 8-bitowy argument bezp. <i>d</i> - 8-bitowe przesunięcie
DJNZ <i>Rr,d</i>	Decrement reg. and jump if reg not zero	$Rr \leftarrow Rr - 1$ i gdy $Rr \neq 0$ to $PC \leftarrow PC + d$	-	<i>Rr</i> - rejestr R0 ÷ R7 <i>d</i> - 8-bitowe przesunięcie
DJNZ <i>ad,d</i>	Decrement addr. contest and jump if not zero	$\langle ad \rangle \leftarrow \langle ad \rangle - 1$ i gdy $\langle ad \rangle \neq 0$ $PC \leftarrow PC + d$	-	<i>ad</i> - bezpośredni adres 8-bitowy <i>d</i> - 8-bitowe przesunięcie
NOP	No operation	<i>Nic nie rób</i>	-	
ACALL <i>adr11</i>	Subroutine call on page		-	<i>adr11</i> - adres 11-bitowy
LCALL <i>adr16</i>	Subroutine call		-	<i>adr16</i> - adres 16-bitowy
RET	Return from subroutine		-	
RETI	Return from interrupt		-	
PUSH <i>ad</i>	Push onto stack	$SP \leftarrow SP + 1$ $(SP) \leftarrow \langle ad \rangle$	-	<i>ad</i> - bezpośredni adres 8-bitowy
POP <i>ad</i>	Pop from stack	$\langle ad \rangle \leftarrow (SP)$ $SP \leftarrow SP - 1$	-	<i>ad</i> - bezpośredni adres 8-bitowy

2. Wybrane dyrektywy asemblera

Linia programu źródłowego ma najczęściej postać:

[etykieta:] mnemonik_rozkazu [operand][,operand][,operand][;komentarz]

ORG 100h – ustawia licznik rozkazów na wskazaną wartość (100h).

END – określenie końca pliku źródłowego.

LICZ EQU 0C2h – przypisanie symbolowi LICZ wartości 0C2h. Symbolowi nie wolno drugi raz nadawać nowej wartości w module.

WSP SET 23 – przypisanie symbolowi WSP wartości 23. Można wielokrotnie nadać temu samemu symbolowi różne wartości.

START CODE 0h – deklaracja adresu w obszarze pamięci programu, nie wolno nadać drugi raz symbolowi nowej wartości.

PORT1 DATA 90h – deklaracja adresu w obszarze pamięci wewnętrznej RAM adresowanej bezpośrednio. Nie wolno nadawać nowej wartości i ma być w przedziale od 0 do 255.

WYNIK1 IDATA 20h – deklaracja adresu pamięci wewnętrznej RAM adresowanej pośrednio, nie wolno nadawać drugi raz wartości symbolowi (wartości z przedziału od 0 do 255).

TEXT_1 XDATA 10 – deklaracja adresu w pamięci zewnętrznej RAM, nie wolno w module nadać drugi raz symbolowi nowej wartości, wyrażenie z przedziału 0-65535.

LED BIT P1.4 – deklaracja adresu bitu w obszarze wewnętrznym RAM lub w obszarze SFR adresowanych bitowo. Nie wolno nadawać drugi raz nowej wartości symbolowi wartości z zakresu 0-255.

Polecenie DB powoduje umieszczenie w kolejnych bajtach pamięci programu wartości wskazanych przez wyrażenie. Wartość wyrażenia z przedziału – 128...255. Przykłady:

TAB: DB 24,24h,1001001B,255 ;od adresu TAB będą umieszczone wartości,

TEXT: DB "to jest apostrof" ;od adresu TEXT umieszczone kody ASCII tego tekstu

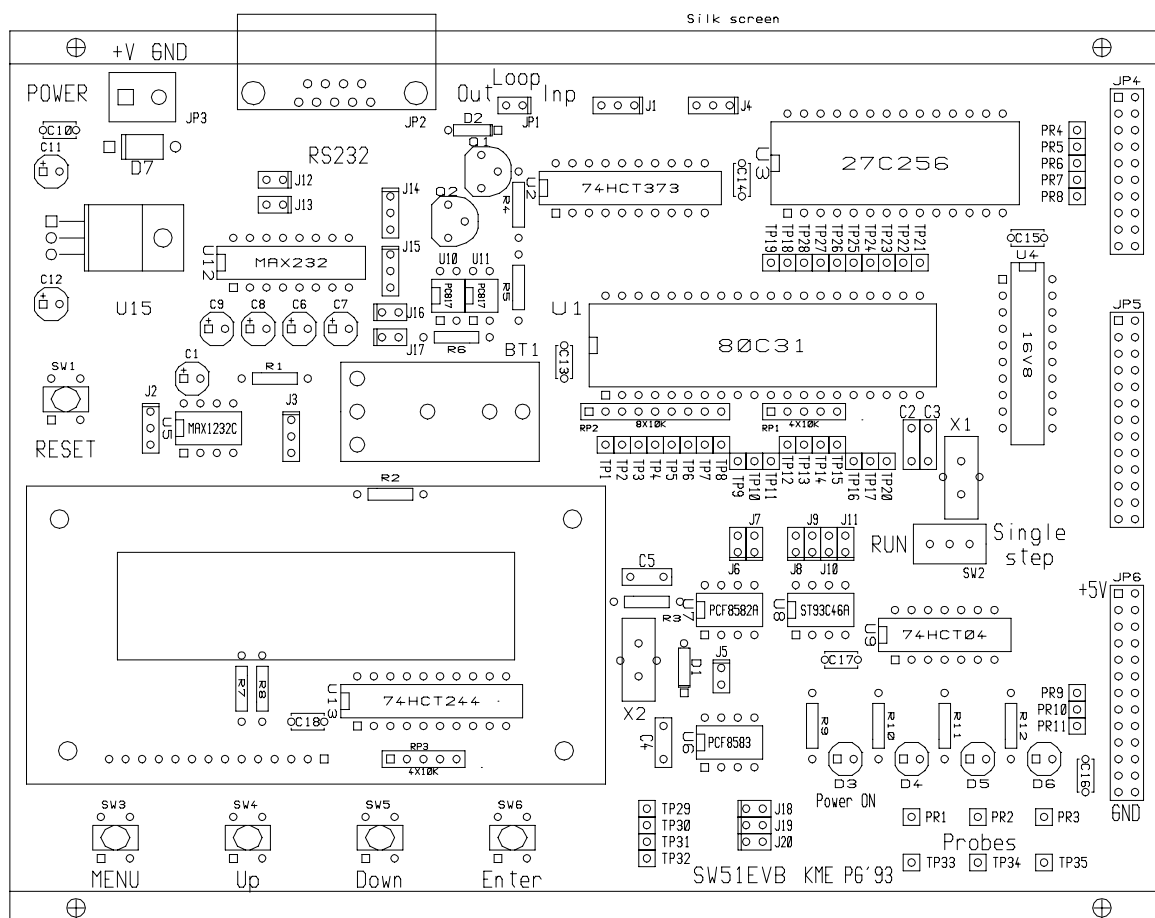
TAB_W: DW 233Ah,'QW' –Tak samo jak DB, lecz wartość wyrażenia z zakresu: 0...65535.

DANE: DS 2 – Rezerwuje w obszarze RAM, wskazanym typem segmentu, liczbę bajtów określoną przez wyrażenie w apostrofie. Wartość wyrażenia apostrofie musi być zdefiniowana w chwili wykonania polecenia.

FLAGA1: DBIT 1 – Rezerwuje liczbę bitów w obszarach adresowanych bitowo określoną przez wyrażenie. Wartość wyrażenia w apostrofie musi być zdefiniowana w chwili wykonania polecenia.

V. Dokumentacja – opis zestawu laboratoryjnego SW51EVB

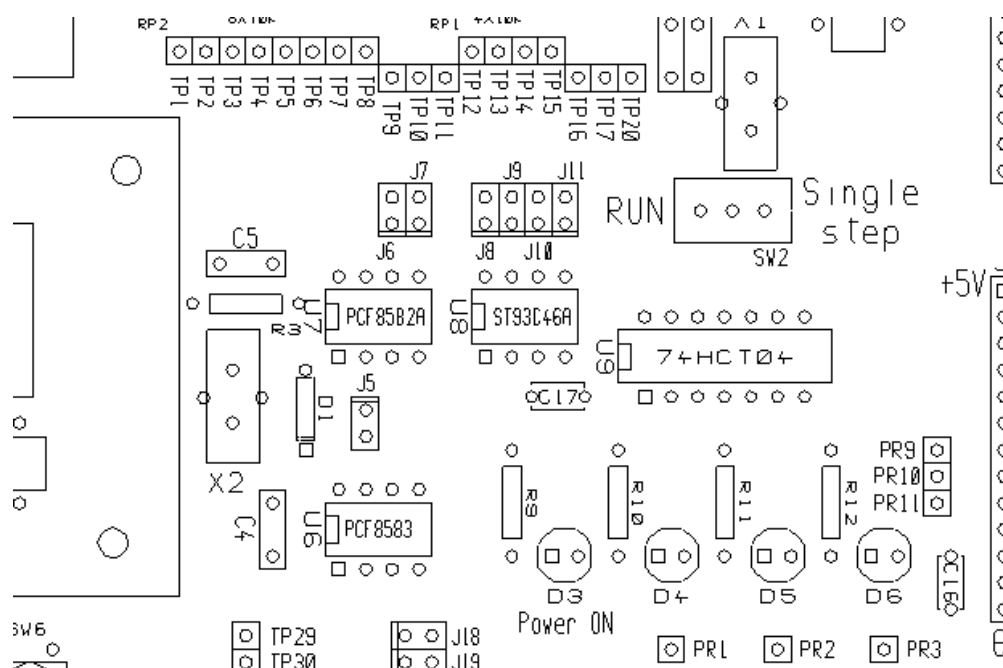
1. Opis ogólny pakietu SW51EVB



Rys. 12. Rozmieszczenie elementów na płycie laboratoryjnej SW51EVB

„Sercem” systemu jest mikrokontroler jednoukładowy U1 typu 80C51 (lub 80C31) pracujący w konfiguracji z zewnętrzną pamięcią programu (wejście EA dołączone do masy - zwora J1 w pozycji 2-3), którą stanowi układ U3 - pamięć EPROM lub (jak w ćwiczeniach symulator pamięci EPROM). Demultipleksowanie portu P0 na szynę danych i młodszą część szyny adresowej dokonuje układ U2 (zatrzask typu 74HCT373). Procesor pracuje z wewnętrznym oscylatorem o częstotliwości ustalonej zewnętrznym kwarcem o częstotliwości 6MHz co daje w efekcie czas trwania cyklu maszynowego 2 μ s.

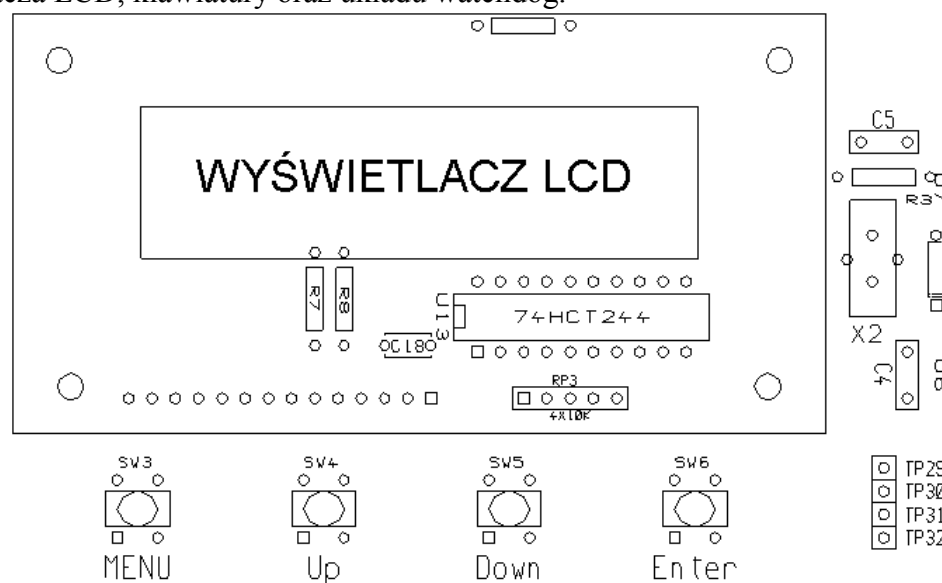
Dla demonstracji możliwości współpracy z różnymi urządzeniami zewnętrznymi wyposażono pakiet w szereg peryferii często wykorzystywanych w praktyce. Jednym z nich jest wyświetlacz LCD 2*16 znaków z wbudowanym kontrolerem. Sposób sterowania poszczególnymi elementami będzie stopniowo opisywany w poszczególnych punktach ćwiczeń jako komentarze do kodu programów.



Rys. 13. Fragment płytki laboratoryjnej SW51EVb z próbnikami PR1 – PR3 i końcówkami TP1 – TP2

Kolejnym elementem wizualizacyjnym są diody LED D4-D6, które poprzez bufory negujące z układu U9 można dołączyć (poprzez próbniki PR1 - PR3) w dowolny punkt pakietu i obserwować jego stan logiczny. Domyślnie próbniki są dołączone do najmłodszych bitów portu P1 (końcówki TP1 - TP3).

Komunikację z użytkownikiem zapewnia klawiatura złożona z przycisków SW3-SW6 oznaczonych jako: MENU, UP, DOWN, ENTER, choć ich faktyczne działanie może być dowolnie zdefiniowane zgodnie z decyzją programisty. Klawiatura jest widoczna jako urządzenie zewnętrzne w obszarze pamięci i jako takiego można odczytać stan. Układ U5 stanowi watchdog. Jeżeli w czasie określonym przez ustawienia zwory J3 nie nastąpi pobudzenie wejścia układu poprzez program użytkownika następuje zerowanie procesora. Układ U4 (GAL16V8) stanowi układ sterujący generujący sygnały niezbędne do podłączenia wyświetlacza LCD, klawiatury oraz układu watchdog.



Rys. 14. Fragment płytki laboratoryjnej SW51EVb z klawiaturą

Dla pokazania możliwości pracy systemowej na pakiecie zaimplementowano 3 rodzaje interfejsu: sprzętowy RS-232 dostępny jako właściwość procesora 51, oraz obsługiwane programowo interfejsy MicroWire oraz I²C. RS-232 jest dostępny na złączu JP2 dołączonym poprzez układ konwersji poziomów (U12 - ICL232) do końcówek RxD i TxD procesora. Interfejs MicroWire pozwala obsługiwać pamięć 93C46 (układ U8), zaś I²C zegar czasu rzeczywistego i pamięć (układy U6 i U7).

Ponieważ na pakiecie nie przewidziano miejsca na zewnętrzną pamięć RAM została ona dołączona poprzez gniazda rozszerzeń JP5 i JP6. Dostępne jest 32KB pamięci, co w zupełności wystarcza do większości zastosowań.

Układy pakietu zasilane są z wbudowanego stabilizatora napięcia (układ U15). Ze względu na ograniczenie wydzielanej mocy w stabilizatorze zaleca się nie przekraczać 10V napięcia zasilającego pakiet. Dioda D7 zabezpiecza układ przed odwrotnym podłączeniem napięcia zasilania.

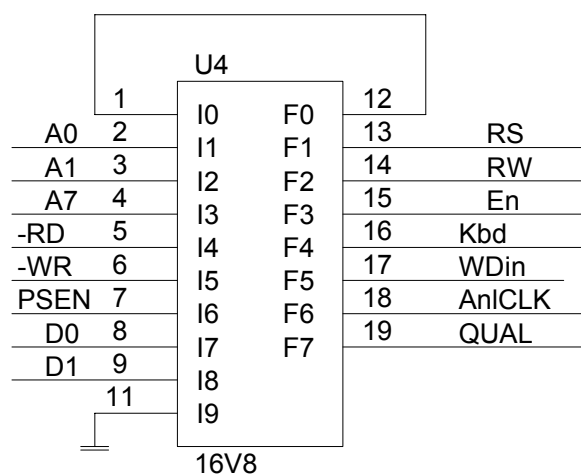
2. Opis poszczególnych bloków pakietu SW51EVB

Znaczenie punktów pomiarowych, próbników i zworek

PR1←→	TP1	wykorzystana dioda D4,
PR2←→	TP2	wykorzystana dioda D5,
PR3←→	TP3	wykorzystana dioda D6,
J1	2-3	zewnętrzna pamięć programu,
J3	1-2	zerowanie po wł. zasilania (bez watchdog'a),
J4	2-3	pamięć 27C64 lub 27C128,
J5	1-2	przerwanie zegarowe 1Hz włączone,
J6,J7	1-2	magistrala I ² C podłączona,
J14,J15	2-3	medium transmisji szeregowej - RS232,
J18,J19,J20	1-2	wyświetlacz podłączony przez U4 (GAL16V8),
pozostałe zworki nie mają znaczenia.		

Uwaga: klawisz RESET znajduje się na płytce rozszerzenia.

Układ GAL16V8



Rys. 15. Schemat układu GAL16V8 z opisem sygnałów sterujących

Na rys. 15 przedstawiono symbol układu GAL16V8 wraz z opisem sygnałów. Dla celów sterowania urządzeniami istotne są następujące sygnały:

RS - sygnał wyboru jednego z dwu wewnętrznych rejestrów wyświetlacza, gdy RS=0 - sterowanie, RS=1 – dane,

RW - sygnał kierunku transmisji:

RW=0 - zapis; RW=1 – odczyt,

En - zezwolenie na zapis/odczyt do/z wyświetlacza,

Kbd - sygnał uaktywniający bufor dołączający klawiaturę do 4 młodszych bitów szyny danych (port P0),

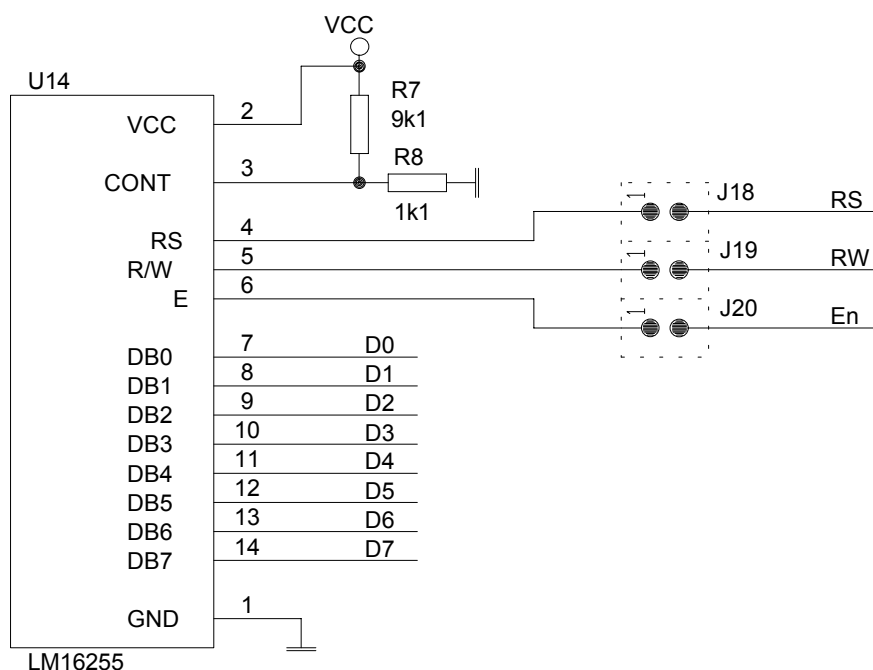
Wdin - sygnał pobudzający układ watchdog - powoduje wyzerowanie wewnętrznego licznika w układzie MAX1232.

Sygnały **AnlCLK** i **QUAL** są wykorzystywane przy podłączaniu analizatora stanów logicznych do złącz rozszerzenia pakietu i nie będą wykorzystywane w ćwiczeniach.

Wymienione sygnały są generowane przez układ U4 realizujący poniższe funkcje:

RS = D0	(stan linii jest przepisywany przy spełnieniu warunku
RW = D1	$A7*/A1*/A0*WR$, czyli zapis pod adres 80H)
$En = A7*/A1*/A0*(WR+RD)$	wyjscie aktywne przy zapisie/odczycie dla adresu 81H
$Kbd = A7*/A1*/A0*RD$	wyjscie aktywne przy odczycie dla adresu 80H
$WDin = A7*/A1*/A0*WR$	wyjscie aktywne przy zapisie dla adresu 82H

Wyświetlacz LCD



Rys. 16. Schemat wyświetlacza LCD

Znaczenie sygnałów RS, RW i En zostało opisane wcześniej przy opisie układu sterującego. Sygnały RS i RW mogą być ustawiane przez linie D0 i D1 przy zapisie do rejestru układu GAL16V8 jako urządzenia w obszarze pamięci pod adresem 80H. Po wstępnym ustawieniu tych linii możliwy jest zapis/odczyt danych do sterownika wyświetlacza - przez zapis/odczyt pamięci zewnętrznej pod adresem 81H (powoduje to uaktywnienie sygnału En).

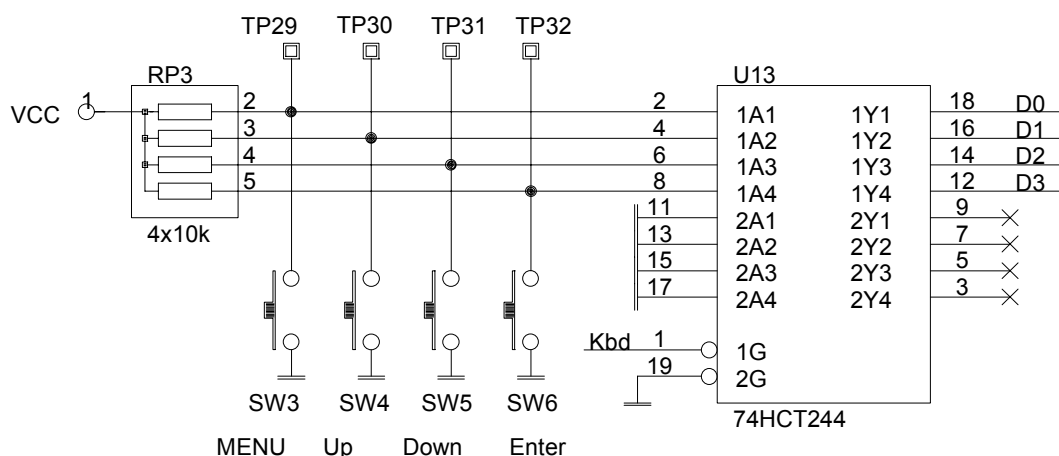
Po każdym zapisie należy sprawdzać flagę gotowości przez odczyt słowa statusu (rejestru sterującego). Znaczenie bitów jest następujące: BF, AC₆, AC₅, AC₄, AC₃, AC₂, AC₁, AC₀

gdzie: BF **busy flag** znacznik zajętości BF=1 zajęty; BF=0 gotowy
 AC₆₋₀ **address counter** wewnętrzny licznik adresujący pamięć wyświetlacza

Tabela. 1. Kody sterujące wyświetlaczem LCD

Nazwa operacji	Kod rozkazu								Opis
	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
Długość słowa	0	0	1	DL	1	0	*	*	Ustawienie długości słowa DL=0 - 4 bity, DL=1 - 8 bitów Operacja musi być wykonana jako 1
Tryb wprowadzania	0	0	0	0	0	1	I/D	S	Określa kierunek przesuwu kursora oraz decyduje czy zawartość wyświetlacza będzie przesuwana po przepełnieniu I/D=0 - wpis na lewo od kursora I/D=1 - wpis na prawo od kursora S=0 - bez przesuwu, S=1 - przesuwana zawartość wyświetlacza
Tryb wyświetlania	0	0	0	0	1	D	C	B	D=0 wyśw. wył.; D=1 wyśw. wł. C=0 kursor wył.; C=1 kursor wł. B=0 bez migotania, B=1 kursor miga
Kasowanie	0	0	0	0	0	0	0	1	Zeruje zawartość wyświetlacza i ustawia kursor na pozycję 00H (pocz. 1 linii)
Powrót kursora	0	0	0	0	0	0	1	*	Ustawia kursor na 00H
Przesuw	0	0	0	1	S/C	R/L	*	*	S/C=0 przesuw wyśw.; S/C=1 przesuw kur. R/L=0 w lewo; R/L=1 w prawo
Ustaw. adresu	1	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	Ustawa wewnętrzny licznik na A ₆₋₀ , kursor na nową pozycję i wpisanie następnego znaku pod wyznaczony adres

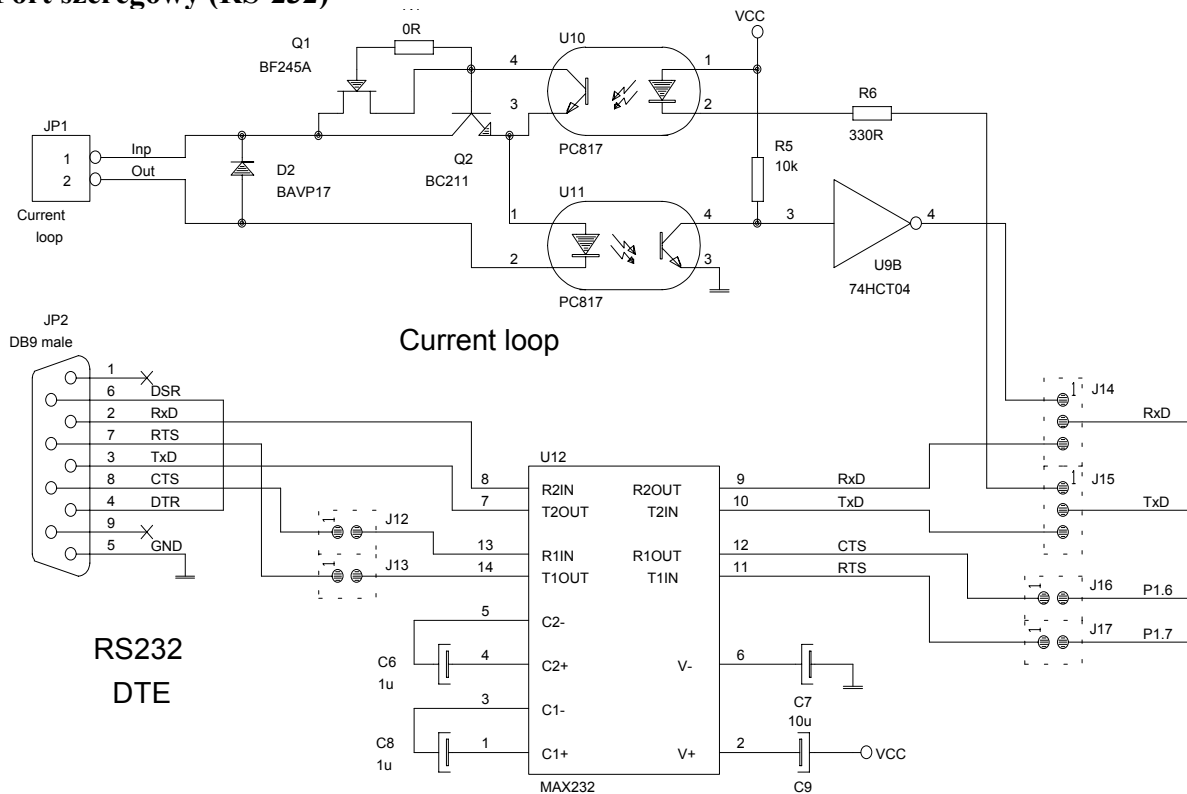
Klawiatura



Rys. 17. Schemat klawiatury

Stan klawiatury może być odczytany przez odczyt rejestru pod adresem 80H w obszarze pamięci zewnętrznej, co powoduje przejście sygnału Kbd w stan aktywny i otwarcie bufora U13. Stan klawiszy odzwierciedlają 4 młodsze bity szyny danych odpowiednio MENU=>D0, Up=>D1, Down=>D2, Enter=>D3. Należy uruchomić poniższy program i obserwować reakcję na naciskanie klawiszy. Zmienić znaki przypisane klawiszom.

Port szeregowy (RS-232)



Rys. 18. Schemat ideowy interfejsu RS232 i pętli prądowej

Mikrokontroler 8051 jest standardowo wyposażony w sterownik transmisji szeregowej zgodnej z RS-232. Jednak dla umożliwienia współpracy z zewnętrznymi urządzeniami w oparciu o ten interfejs konieczna jest konwersja poziomów z TTL na poziomy RS-232. To zadanie wykonuje układ U12 (MAX232). Na pakiecie przewidziano również alternatywny nośnik transmisji szeregowej jakim jest pętla prądowa (*current loop*). Wyboru medium transmisyjnego dokonujemy za pomocą zworek J14 i J15.

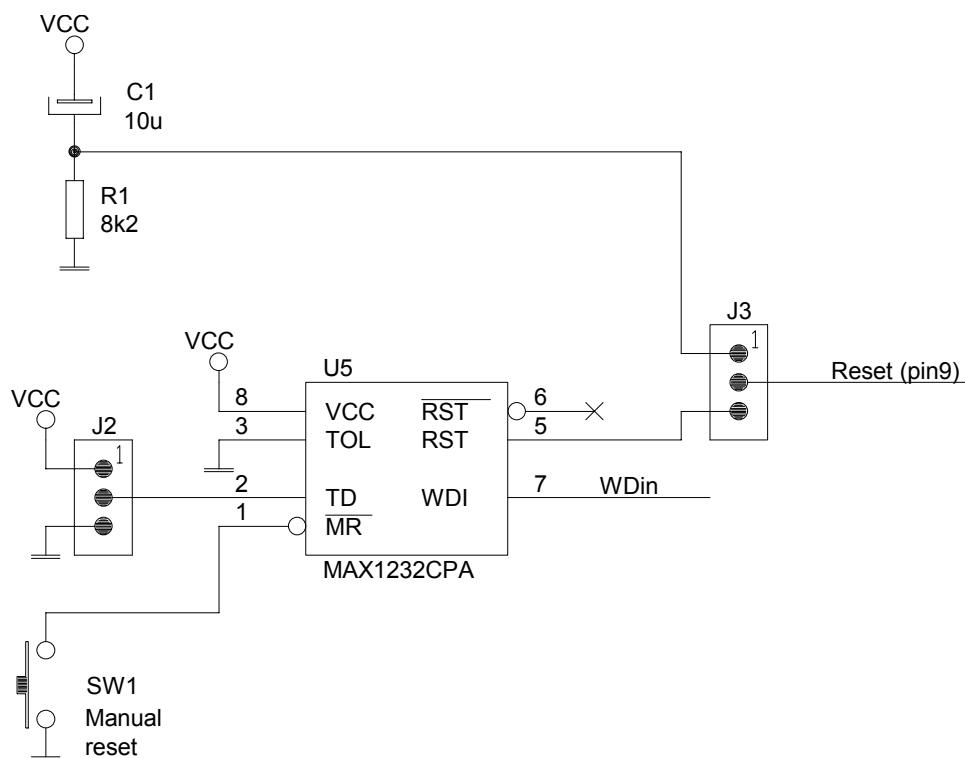
Do współpracy poprzez RS-232 wykorzystamy komputer PC pracujący jako terminal.

Układ nadzorujący - watchdog

Układ nadzorujący watchdog najczęściej spełnia rolę stróża w systemie mikroprocesorowym. W systemie uruchomieniowym SW51EVB pełni 3 funkcje:

- generuje sygnał reset po włączeniu i każdym spadku napięcia zasilania (a więc pełni rolę monitora zasilania),
- generuje sygnał reset po każdym naciśnięciu klawisza **Manual reset**,
- posiada wbudowany wewnętrzny oscylator i licznik. Licznik ten powinien być cyklicznie zerowany poprzez wejście WDI, w przeciwnym razie po każdym przepełnieniu licznika nastąpi wygenerowanie sygnału reset. Minimalny czas pomiędzy pobudzeniami licznika jest zależny od częstotliwości oscylatora a ta zależy od ustawień zworki J2:

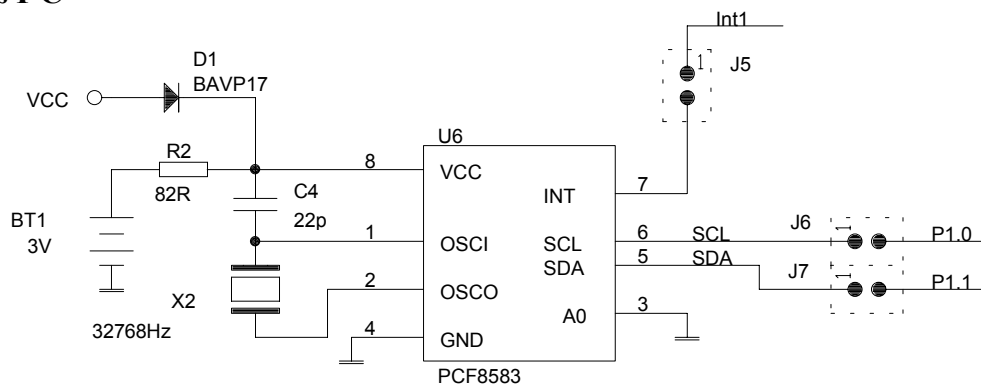
J2	brak	0.6s
J2	1-2	1.2s
J2	2-3	0.15s



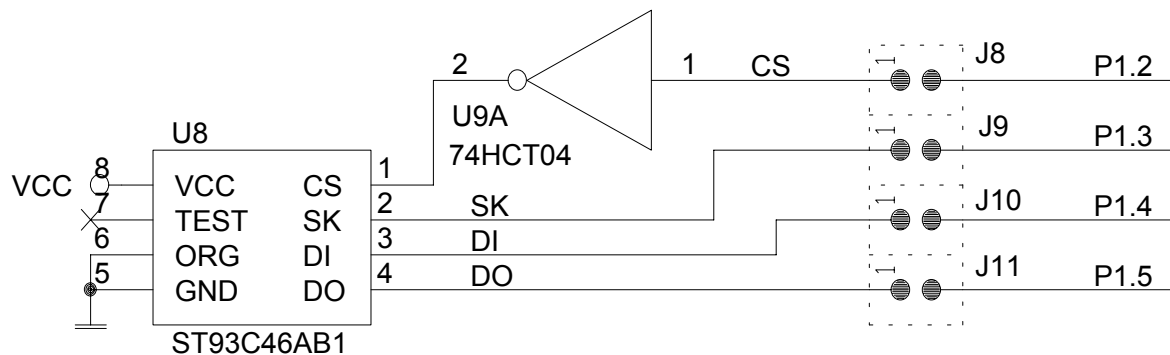
Rys. 19. Schemat ideowy układu Watchdog

Uwaga: Aby włączyć układ watchdog należy zworę J3 przełożyć w pozycję 2-3!

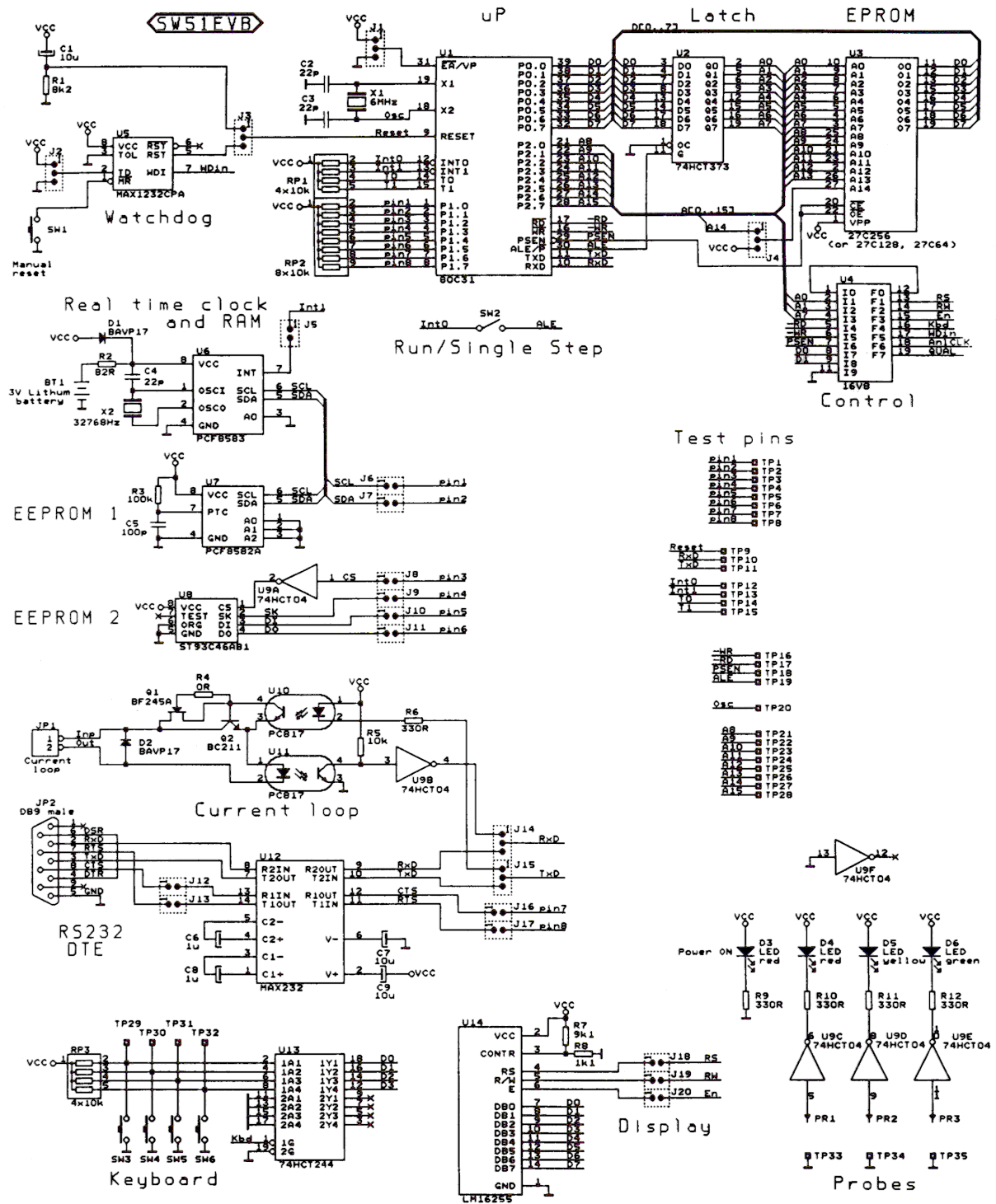
Interfejs I²C

Rys. 20. Schemat ideowy układu zegara czasu rzeczywistego z interfejsem I²C

Interfejs Microwire



Rys. 21. Schemat ideowy pamięci EEPROM z interfejsem Microwire



Rys. 22. Schemat ideowy pakietu SW51EVB

VI. Dokumentacja – kody źródłowe przykładowych programów

1. Ćwiczenie 5

a. Plik **przyk1.asm**

```
; Przykład pierwszy - obsługa portu P1 mk 80C51/52
; Laboratorium Mikrokontrolery i Mikrosystemy

$mod52

ORG 000H

BEGIN:
    clr P1.0      ;zgas diode D4 (wyzeruj linię P1.0)
    clr P1.1      ;zgas diode D5 (wyzeruj linię P1.1)
    clr P1.2      ;zgas diode D6 (wyzeruj linię P1.2)

    acall ZWLOKA  ;wygeneruj opoznienie (podprogram)

    setb P1.0     ;zapal diode D4 (ustaw linię P1.0)
    setb P1.1     ;zapal diode D5 (ustaw linię P1.1)
    setb P1.2     ;zapal diode D6 (ustaw linię P1.2)

    acall ZWLOKA

    jmp BEGIN     ; na początek

ZWLOKA:
    mov R1,#194   ;generacja opóznienia
                  ;do rejestru R1 załaduj wartosc 194
SKOK1:
    mov R0,#255   ;do rejestru R0 załaduj wartosc 255
    djnz R0,$      ;dekrementuj zawartosc R0 i jesli R0<>0
                  ;skocz do adresu $ - adres aktualnego rozkazu
                  ;czyli skocz do tego samego miejsca
                  ;zatem ten rozkaz bedzie powtorzony 255 razy
    djnz R1,SKOK1 ;dekrementuj zawartosc R1 i jesli R1<>0
                  ;powtorz petle z dekrementacją R0
    ret
END
```

b. Plik **przyk2.asm**

```
; Przykład drugi - przekazywanie parametrów do podprogramu mk 80C51/52
; Laboratorium Mikrokontrolery i Mikrosystemy

$mod52

ORG 000H

    jmp BEGIN

ORG 100H
BEGIN:
    clr P1.0      ;zgas diode D4 (wyzeruj linię P1.0)
    setb P1.1     ;zapal diode D5 (ustaw linię P1.1)
    clr P1.2      ;zgas diode D6 (wyzeruj linię P1.2)

    mov R3,#8     ;zadajemy opoznienie (8 jednostki)
    call ZWLOKA   ;wygeneruj opoznienie (podprogram)

    setb P1.0     ;zapal diode D4 (ustaw linię P1.0)
    clr P1.1      ;zgaś diode D5 (wyzeruj linię P1.1)
```

```

        setb P1.2      ;zapal diode D6 (ustaw linie P1.2)

        mov R3,#2      ;zadajemy opoznienie (2 jednostki)
        call ZWLOKA    ;wygeneruj opoznienie (podprogram)

        jmp BEGIN

ZWLOKA:      ;podprogram generujący opóźnienie
        mov R1,#255
Z1:          mov R0,#196
            djnz R0,$
            djnz R1,Z1
            djnz R3,ZWLOKA
            ret

END

```

c. Plik przyk3.asm

; Przykład trzeci - praca na bajtach mk 80C51/52
; Laboratorium Mikrokontrolery i Mikrosystemy

\$mod52

```

;Definicje stalych (slowo kluczowe 'equ')
; Sufiksy przy liczbach
; 'B' - liczba binarna
; 'H' - liczba hexalna
; 'O' - liczba osemkowa
; 'D' lub brak - liczba dziesietna

```

```

LEDS_ON equ 00000111B
LEDS_OFF equ 11111000B
LED1_ON equ 00000001B
LED2_ON equ 00000010B
LED3_ON equ 00000100B
OPU1 equ 5
OPU2 equ 1

```

```

ORG 000H
    jmp BEGIN

```

ORG 100H

```

BEGIN:
    orl P1,#LEDS_ON      ;zapal diody
    mov R3,#OPU1         ;zadajemy opoznienie
    call ZWLOKA          ;wygeneruj opoznienie

PETLA:
    anl P1,#LEDS_OFF     ;zgas diody
    mov R3,#OPU2         ;zadajemy opoznienie
    call ZWLOKA          ;wygeneruj opoznienie

    orl P1,#LED1_ON      ;zapal diode D4
    mov R3,#OPU2         ;zadajemy opoznienie
    call ZWLOKA          ;wygeneruj opoznienie

    orl P1,#LED2_ON      ;zapal diode D5
    mov R3,#OPU2         ;zadajemy opoznienie
    call ZWLOKA          ;wygeneruj opoznienie

    orl P1,#LED3_ON      ;zapal diode D6

```



```

        mov R3,#OPU2           ;zadajemy opoznienie
        call ZWLOKA           ;wygeneruj opoznienie

        jmp PETLA

ZWLOKA:                               ;Podprogram generujący opóźnienie
        mov R1,#255

Z1:
        mov R0,#196
        djnz R0,$
        djnz R1,Z1
        djnz R3,ZWLOKA
        ret

END

```

d. Plik **przyk4.asm**

; Przykład czwarty - wykorzystanie stosu mk 80C51/52
; Laboratorium Mikrokontrolery i Mikrosystemy

\$mod52

```

ZGAS    equ  07H           ;maska
OPU     equ  3             ;stała opoznienia

```

```

ORG 000H
START:
        jmp BEGIN

```

ORG 100H

```

BEGIN:
        mov B,#ZGAS        ;w rej. B maska wyłączająca diody
        clr A
        mov SP,#30H        ;wskaźnik stosu ustawiony na 30H

PETLA:
        inc A               ;inkrementacja A i przygotowanie
        anl A,B             ;do wyświetlenia wyniku
        cpl A

        mov P1,A            ;wyświetlenie wyniku
        mov R3,#OPU        ;zadajemy opoznienie
        call ZWLOKA        ;wygeneruj opoznienie

        jmp PETLA

ZWLOKA:                               ;Podprogram generujący opóźnienie
        push B              ;zawartosc rejestru B na stos

        push ACC            ;zawartosc akumulatora na stos
        mov B,#255

Z1:
        mov A,#196
        djnz ACC,$
        djnz B,Z1
        djnz R3,ZWLOKA

        pop ACC             ;odzyskaj zawartosc akumulatora ze stosu
        pop B              ;odzyskaj zawartosc rejestru B ze stosu

        ret

```

END

e. Plik **przyk5.asm**

; Przykład piąty - obsługa przerwań z licznika T0 mk 80C51/52
 ; Laboratorium Mikrokontrolery i Mikrosystemy

```
$mod52

LED equ P1.0
T0MODE equ 01h          ;timer 0 jako 16-bitowy licznik

ORG 000h

START:
    jmp BEGIN

                                ;Wektory przerwan:
ORG 003h                    ;przerwanie zewnetrzne INT0

ORG 00Bh                    ;przerwanie od timer'a T0
    ajmp OBST0              ;skok do procedury obsługi przerwania

ORG 013h                    ;przerwanie zewn. INT1

ORG 01Bh                    ;przerwanie od timer'a T1

ORG 023h                    ;przerwanie od portu szeregowego

ORG 0100H

BEGIN:
    mov SP,#30H             ;wskaźnik stosu ustawiony na 30H
    mov TMOD,#T0MODE        ;ustawienie trybu licznika 0
    setb TR0                 ;start timer'a 0
    setb ET0                 ;zezwolenie na przyjmowanie przerwan timer'a 0
    setb EA                  ;uaktywnienie wszystkich przerwan

    sjmp $                  ;pętla główna programu - oczekiwanie na przerwania

OBST0:
    cpl LED                 ;zmien stan diody
    reti                   ;powrot z obsługi przerwania

END
```

f. Plik **przyk6.asm**

; Przykład szósty - odczytywanie precyzyjnych odcinków
 ; czasu licznikiem T0 mk 80C51/52
 ; Laboratorium Mikrokontrolery i Mikrosystemy

```
$mod52

T0MODE equ 00000001B        ;timer 0 jako 16-bitowy licznik
CLOCKH equ 09EH             ;przerwanie co 50ms przy f=6MHz
CLOCKL equ 058H             ;bo 65536-0.05s/2us=40536d=9e58h
LED equ P1.0

ORG 000h

START:
    jmp BEGIN

                                ;Wektory przerwan:
ORG 003h                    ;przerwanie zewnetrzne INT0
```

```

ORG 00Bh          ;przerwanie od timer'a T0
    ajmp OBST0    ;skok do procedury obsługi przerwania

ORG 013h          ;przerwanie zewn. INT1

ORG 01Bh          ;przerwanie od timer'a T1

ORG 023h          ;przerwanie od portu szeregowego

ORG 0100H         ;przerwanie od portu szeregowego

BEGIN:
    mov SP,#30H   ;wskaźnik stosu ustawiony na 30H
    mov TMOD,#T0MODE ;ustawienie trybu licznika 0
    mov TH0,#CLOCKH
    mov TL0,#CLOCKL ;wartosci poczatkowe licznika 0
    setb TR0        ;start timer'a 0
    setb ET0        ;zezwozenie na przyjmowanie przerw timer'a 0
    setb EA         ;uaktywnienie wszystkich przerw

sjmp $            ;pętla główna programu - oczekiwanie na przerwanie

OBST0:
    mov TH0,#CLOCKH
    orl TL0,#CLOCKL ;uwzgledniajac, fakt, ze licznik caly czas liczy
                    ;dlatego nalezy dodac to co licznik zdazyl
                    ;juz naliczyc (dlatego 'orl')
    cpl LED         ;zmien stan diody

    reti

END

```

g. Plik **przyk7.asm**

; Przykład siódmy - obsługa przerwania zewnętrznego mk 80C51/52
; Laboratorium Mikrokontrolery i Mikrosystemy

\$mod52

```

T0MODE equ 01H    ;timer 0 jako 16-bitowy licznik
CLOCKH equ 09EH   ;przerwanie co 50ms przy f=6MHz
CLOCKL equ 058H   ;bo 65536-0.05s/2us=40536d=9e58h

ZGAS equ 00000111B
ZM equ 20H        ;zmienna pomocnicza

ORG 000h

START:
    jmp BEGIN

ORG 003h          ;Wektory przerw:
    ajmp OBSINT0  ;przerwanie zewnetrzne INTO

ORG 00Bh          ;przerwanie od timer'a T0
    ajmp OBST0    ;skok do procedury obsługi przerwania

ORG 013h          ;przerwanie zewn. INT1

ORG 01Bh          ;przerwanie od timer'a T1

ORG 023h          ;przerwanie od portu szeregowego

ORG 0100H         ;przerwanie od portu szeregowego

```

```

BEGIN:
    mov B,#ZGAS          ;w rej. B maska wyłączająca diody
    clr A
    mov SP,#30H          ;wskaźnik stosu ustawiony na 30H

    mov TMOD,#T0MODE     ;ustawienie trybu licznika 0
    mov TH0,#CLOCKH
    mov TL0,#CLOCKL      ;wartosci początkowe licznika 0
    setb TR0             ;start timer'a 0
    setb ET0             ;zezwozenie na przyjmowanie przerwan timer'a 0
    setb EX0             ;zezwozenie na przerwanie od INT0
    setb IT0             ;przerwanie INT0 aktywne zboczem opadającym
    setb EA              ;uaktywnienie wszystkich przerwan

    sjmp $               ;pętla główna programu - oczekiwanie na przerwania

OBSINT0:
    cpl ZM              ;negacja flagi ZM
    reti

OBST0:
    mov TH0,#CLOCKH
    orl TL0,#CLOCKL      ;uwzgledniajac, fakt, ze licznik caly czas liczy
                        ;dlatego nalezy dodac to co licznik zdazyl
                        ;juz naliczyc (dlatego 'orl')

    jb ZM,SKOK1
    inc A                ;inkrementacja A
    ajmp SKOK2

SKOK1:
    dec A                ;dekrementacja A

SKOK2:
    anl A,B              ;przygotowanie do wyświetlenia wyniku
    cpl A
    mov P1,A             ;wyświetlenie wyniku

    reti
END

```

2. Ćwiczenie 6

a. Plik fwysw.asm

; Laboratorium Mikrokontrolery i Mikrosystemy
; Zespół stałych i funkcji obsługi wyświetlacza LCD

```

LCDCTRL equ 080h        ;adres rejestru sterujacego (GAL16V8)
LCDADDR equ 081h        ;adres rejestru danych wyswietlacza
DATABUFF equ 059H       ;adres bufora znaku do wysw. LCD
DISPADDR equ 05AH       ;adres w RAM wysw. LCD do zapisu
LCDRS equ Acc.0          ;wybor rejestru wysw. 0=sterujacy; 1=dane
LCDRW equ Acc.1          ;wybor kierunku trans. 0=zapis; 1=odczyt

                        ;słowa programujące wyświetlacz:
LCDDL equ 38H            ;zaprogramowanie dl. słowa danych (8-bit) - LCD Data Length
LCDEMS equ 06H           ;tryb wprowadzania: inkrementacja, - LCD Entry Mode Set
LCDDC equ 0CH            ;display control: on, cursor off, unblinks - LCD Display Control

                        ;słowa sterujące wyświetlaczem:
LCDCLR equ 01H           ;kasowanie wysw.
LCDHOME equ 02H          ;kursor na początek (ADRES 0H)

```

; Podprogram INITLCD - Inicjalizacja wyświetlacza LCD

INITLCD:

```

mov DATABUFF,#LCDDL      ;słowo programujące długość słowa
clr LCDRS                 ;ustaw rejestr sterujący
call WRITELCD             ;zapisz do rejestru wysw.
call CHECKLCD             ;czekaj aż gotowe
mov DATABUFF,#LCDEMS     ;tryb wprowadzania danych do wysw.
clr LCDRS                 ;ustaw rejestr sterujący
call WRITELCD             ;zapisz
call CHECKLCD             ;gotowe?
mov DATABUFF,#LCDDC      ;sposób wyświetlania
clr LCDRS                 ;rejestr sterujący
call WRITELCD             ;zapisz
call CHECKLCD             ;gotowe?
mov DATABUFF,#LCDCLR     ;czyszczenie wyświetlacza
clr LCDRS                 ;rejestr sterujący
call WRITELCD             ;zapisz
ret

```

; Podprogram CHECKLCD - sprawdza stan flagi BUSY i czeka na gotowość

CHECKLCD:

```

setb LCDRW                ;odczyt
clr LCDRS                 ;rejestru sterującego (status)
mov R0,#LCDCTRL           ;ustaw adres GAL16V8
movx @R0,A                ;zapisz nowe stany RS i R/W
mov R0,#LCDADDR           ;ustaw adres wysw.
movx A,@R0                ;odczytaj słowo statusu
jb ACC.7,CHECKLCD         ;sprawdź BF (bit 7) skocz jeśli nie gotowe
ret

```

; Podprogram WRITELCD - zapisuje dane lub słowo sterujące do wysw.

WRITELCD:

```

clr LCDRW                ;zapis
mov R0,#LCDCTRL          ;adres GAL16V8
movx @R0,A                ;zapisz
mov A,DATABUFF            ;pobierz dane do zapisania
mov R0,#LCDADDR           ;adres wysw.
movx @R0,A                ;zapisz
ret

```

; Podprogram WRITETX -

```

;
;
;

```

wypisuje tekst na wysw.
adres tekstu powinien znaleźć się w DPTR
tekst zakończony zerem

WRITETX:

```

call CHECKLCD             ;sprawdź czy LCD gotowy
mov A,DISPADDR            ;pobierz adres komórki wysw. dla 1 znaku
orl A,#80H                ;przygotuj kod rozkazu ustawiającego adr.
mov DATABUFF,A            ;prześlij jako słowo sterujące do wysw.
clr LCDRS                 ;wybierz rejestr ster.
call WRITELCD             ;zapisz

```

WRITELP:

```

call CHECKLCD             ;gotowe?
clr A                     ;pobierz 1 znak do wypisania
movc A,@A+DPTR            ;jesli nie ostatni znak to dalej
jnz WRITECH
ret

```

WRITECH:

```

mov DATABUFF,A            ;prześlij do bufora
setb LCDRS                ;rejestr danych wysw.
call WRITELCD             ;zapisz
inc DISPADDR              ;następna pozycja wysw.
inc DPTR                  ;następny znak
jmp WRITELP

```

b. Plik **fklaw.asm**

```
; Laboratorium Mikrokontrolery i Mikrosystemy
; Zespół stałych i funkcji obsługi klawiatury
```

```
KBD equ 080h           ;adres bufora klawiatury
RPKEY equ 05CH          ;ostatnio odczytany klawisz
KBDMASK equ 00Fh        ;maska klawiatury
RPD equ 8               ;opoznienie trybu autopowtarzania w 0.1s
ARPD equ 1              ;czas autopowtarzania w 0.1s
RPDELAY equ 05DH        ;licznik odmierzający czas wejścia w tryb autopowt.
ARPDELAY equ 05EH        ;licznik autopowtarzania
ARM equ 009H            ;znacznik trybu autopowtarzania
```

```
;      D7                D0
;      +-----+
;      | 0 | 0 | 0 | 0 | E | D | U | M |
;      +-----+
;      U : Up      D : Down
;      M : Menu    E : Enter
```

```
MENUKEY equ ACC.0
UPKEY equ ACC.1
DOWNKEY equ ACC.2
ENTERKEY equ ACC.3
CHARM equ 'M'
CHARU equ 'U'
CHARD equ 'D'
CHARE equ 'E'
```

```
; Podprogram READKBD   odczytuje stan klawiaturki
```

```
READKBD:
    mov R0,#KBD          ;ustaw adres klawiatury
    movx A,@R0            ;odczytaj stan
    anl A,#KBDMASK        ;wytnij 4 starsze bity
    mov R2,A              ;zapamiętaj
    mov R3,#0FFH
    djnz R3,$             ;odczeka 400us
    movx A,@R0            ;ponownie odczytaj stan klawiatury
    anl A,#KBDMASK
    xrl A,R2               ;porównaj z poprzednim odczytem
    jnz NOKEYPR           ;jeśli różne to błąd
    mov A,R2               ;w przeciwnym razie odtwórz stan
    cpl A                 ;neguj
    mov R2,A              ;przechowaj kod

                                ;analiza wcisniętych klawiszy
    mov R3,#4
    mov R4,#0
KBDLP1:
    jnb ACC.0,NOKEY       ;kolejno sprawdzaj czy wcisnięty
    inc R4
NOKEY:
    rr A
    djnz R3,KBDLP1        ;R4 zawiera ilość wcisniętych klawiszy
    mov A,R4
    jz NOKEYPR            ;jeśli 0 żaden klawisz nie wcisnięty
    dec A
    jnz NOKEYPR           ;jeśli więcej niż jeden pomin
    mov A,RPKEY           ;poprzednio wcisnięty klawisz
    xrl A,R2               ;porównaj z aktualnym
    jnz NEWKEY            ;jeśli nowy aktualizuj
    jb ARM,ARMMODE        ;skocz jeśli już tryb powtarzania
    mov A,RPDELAY         ;czas powtórzenia
    jnz NRPYET            ;skocz jeśli niedość
    setb ARM              ;jeśli dość ustaw znacznik trybu powt.
```

```

        mov ARPDELAY,#ARPD ;wczytaj czas powtarzania
        mov A,R2           ;kod klawisza do akumulatora
        ret
NRPYET:
        clr A
        ret
ARMMODE:
        mov A,ARPDELAY
        jnz NRPYET         ;jesli nie czas na nast. powtorzenie
        mov ARPDELAY,#ARPD
        mov A,R2
        ret
NARPYET:
        clr A
        ret
NEWKEY:
        clr ARM             ;jesli nowy klawisz wytlacz tryb powt.
        mov RPDELAY,#RPD
        mov A,R2
        mov RPKEY,A        ;zapamietaj nowy klawisz
        ret
NOKEYPR:
        mov RPKEY,#0
        clr A
        clr ARM
        ret

```

3. Ćwiczenie 7

a. Plik **przyk1.c**

```

/* Przykład pierwszy - prosty program w języku C */
/* Laboratorium Mikrokontrolery i Mikrosystemy */

#include <reg51.h> /* deklaracje SFR, portów, bitów itp. */

/* zatrzymanie wykonywania programu na czas <del> podany w 0.1s */

void delay(unsigned char del)
{
    unsigned int i,j;

    for (j=0; j < del; j++) for (i=0; i < 3000; i++);
}

/* początek programu głównego */

void main(void)
{
    P1_0=1;      /* bit 0 portu P1 ustaw na 1 */
    P1_1=0;      /* bit 1 portu P1 ustaw na 0 */
    P1_2=1;      /* bit 2 portu P1 ustaw na 1 */

    while (1)    /* petla nieskonczona */
    {
        P1_0=0;
        P1_1=1;
        P1_2=0;
        delay(5); /* opoznienie 0.5 sek. */
        P1_0=1;
        P1_1=0;
        P1_2=1;
        delay(5);
    }
}

```

b. Plik **reg51.h**

```
/* (c) Copyright KEIL ELEKTRONIK GmbH. 1989, All rights reserved. */
/* Register Declarations for 8051 Processor */
```

```
/* BYTE Register */
```

```
sfr P0 = 0x80;
sfr P1 = 0x90;
sfr P2 = 0xA0;
sfr P3 = 0xB0;
sfr PSW = 0xD0;
sfr ACC = 0xE0;
sfr B = 0xF0;
sfr SP = 0x81;
sfr DPL = 0x82;
sfr DPH = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0 = 0x8A;
sfr TL1 = 0x8B;
sfr TH0 = 0x8C;
sfr TH1 = 0x8D;
sfr IE = 0xA8;
sfr IP = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;
```

```
/* BIT Register */
/* PSW */
```

```
sbit CY = 0xD7;
sbit AC = 0xD6;
sbit F0 = 0xD5;
sbit RS1 = 0xD4;
sbit RS0 = 0xD3;
sbit OV = 0xD2;
sbit P = 0xD0;
```

```
/* TCON */
```

```
sbit TF1 = 0x8F;
sbit TR1 = 0x8E;
sbit TF0 = 0x8D;
sbit TR0 = 0x8C;
sbit IE1 = 0x8B;
sbit IT1 = 0x8A;
sbit IE0 = 0x89;
sbit IT0 = 0x88;
```

```
/* IE */
```

```
sbit EA = 0xAF;
sbit ES = 0xAC;
sbit ET1 = 0xAB;
sbit EX1 = 0xAA;
sbit ET0 = 0xA9;
sbit EX0 = 0xA8;
```

```
/* IP */
```

```
sbit PS = 0xBC;
sbit PT1 = 0xBB;
sbit PX1 = 0xBA;
sbit PT0 = 0xB9;
sbit PX0 = 0xB8;
```

```
/* P3 */
```

```
sbit RD = 0xB7;
sbit WR = 0xB6;
```

```
sbit T1 = 0xB5;
sbit T0 = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD = 0xB1;
sbit RXD = 0xB0;
```

```
/* P1 */
```

```
sbit P1_0 = 0x90;
sbit P1_1 = 0x91;
sbit P1_2 = 0x92;
sbit P1_3 = 0x93;
sbit P1_4 = 0x94;
sbit P1_5 = 0x95;
sbit P1_6 = 0x96;
sbit P1_7 = 0x97;
```

```
/* P3 */
```

```
sbit P3_0 = 0xB0;
sbit P3_1 = 0xB1;
sbit P3_2 = 0xB2;
sbit P3_3 = 0xB3;
sbit P3_4 = 0xB4;
sbit P3_5 = 0xB5;
sbit P3_6 = 0xB6;
sbit P3_7 = 0xB7;
```

```
/* SCON */
```

```
sbit SM0 = 0x9F;
sbit SM1 = 0x9E;
sbit SM2 = 0x9D;
sbit REN = 0x9C;
sbit TB8 = 0x9B;
sbit RB8 = 0x9A;
sbit TI = 0x99;
sbit RI = 0x98;
```

```
/* EEPROM */
```

```
sfr EADRL = 0xF2;
sfr EADRH = 0xF3;
sfr EDATA = 0xF4;
sfr ETIM = 0xF5;
sfr ECNTRL = 0xF6;
```