
Laboratorium systemów mikroprocesorowych



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



UMCS
UNIWERSYTET MEDYCZY I ŻYWIENIA
W LUBLINIE

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt „Programowa i strukturalna reforma systemu kształcenia na Wydziale Mat-Fiz-Inf”.
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Człowiek-najlepsza inwestycja

UNIwersYTET MARIi CURIE-SKŁODOWSKIEJ
WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI
INSTYTUT INFORMATYKI

Laboratorium systemów mikroprocesorowych

Jerzy Kotliński
Sławomir Kotyra



LUBLIN 2011

Instytut Informatyki UMCS

Lublin 2011

Jerzy Kotliński, Sławomir Kotyra

LABORATORIUM SYSTEMÓW MIKROPROCESOROWYCH

Recenzent: Elżbieta Ratajewicz-Mikołajczak

Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach
Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach
Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl

Wydawca

Uniwersytet Marii Curie-Skłodowskiej w Lublinie

Instytut Informatyki

pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin

Redaktor serii: prof. dr hab. Paweł Mikołajczak

www: informatyka.umcs.lublin.pl

email: dyrii@hektor.umcs.lublin.pl

Druk

ESUS Agencja Reklamowo-Wydawnicza Tomasz Przybylak

ul. Ratajczaka 26/8

61-815 Poznań

www: www.esus.pl

ISBN: 978-83-62773-04-6

SPIS TREŚCI

PRZEDMOWA	VII
CZEŚĆ 1	1
O BUDOWIE I DZIAŁANIU MIKROKOMPUTERÓW.....	1
1.1. WPROWADZENIE.....	2
1.2. BUDOWA I DZIAŁANIE MIKROKONTROLERÓW RODZINY MSC-51.	5
1.3. PRACOWNIA SYSTEMÓW MIKROPROCESOROWYCH.	63
CZEŚĆ 2	99
O PROGRAMOWANIU MIKROKOMPUTERÓW.....	99
2.1. ŚRODOWISKO MIKROKOMPUTERA I JEGO OBSŁUGA.	100
2.2. PROGRAMOWANIE MIKROKONTROLERÓW.	121
CZEŚĆ 3	141
UZUPEŁNIENIA	141
3.1. TABELI I OPISY.....	142
3.2. SKRÓCONA LISTA ROZKAZÓW.....	159
3.3. PEŁNA LISTA ROZKAZÓW.....	165
BIBLIOGRAFIA.....	217
SKOROWIDZ.....	219

PRZEDMOWA

Niniejszy skrypt został opracowany jako pomoc dydaktyczna dla studentów Instytutu Informatyki UMCS w Lublinie. Skrypt jest przeznaczony dla studentów wykonujących ćwiczenia w laboratorium systemów mikroprocesorowych. Zajęcia w tym laboratorium są związane z programowaniem małych systemów mikroprocesorowych.

Skrypt został podzielony na 3 części. Część 1 jest poświęcona omówieniu architektury wybranych mikrokontrolerów rodziny MCS-51. W części tej omówiono również poszczególne zadania, które powinny być wykonane w laboratorium.

Część 2 skryptu jest poświęcona sposobom programowej obsługi sprzętu mikroprocesorowego. W części tej określono pojęcie środowiska systemu mikroprocesorowego i omówiono problemy związane z obsługą zdarzeń w tym środowisku. W części 2 skryptu podano wskazówki umożliwiające rozpoczęcie programowania mikrokomputerów w języku assemblera i C.

W części 3 skryptu zgromadzono dodatkowe informacje, które mogą być użyteczne w trakcie projektowania i pisania programów. Oprócz dodatkowych, stabelaryzowanych opisów urządzeń I/O, w części 3 skryptu umieszczono skróconą i pełną listę rozkazów mikrokontrolera z rodziny MCS-51.

Podstawowym źródłem bibliograficznym, umożliwiającym dokładny opis struktury mikrokontrolera z rodziny MCS-51, była publikacja katalogowa f-my INTEL [1]; materiałami pomocniczymi były opracowania [6, 7, 17].

Podrozdział 1.2.2 w części II skryptu, "Programowanie w języku C", opracował Sławomir Kotyra. Pozostałe elementy skryptu opracował Jerzy Kotliński.

Wszystkie nazwy własne i znaki towarowe użyte w niniejszej publikacji są własnością odpowiednich firm.

CZĘŚĆ 1

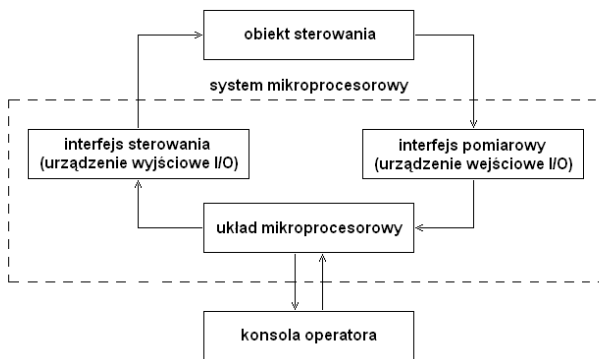
O BUDOWIE I DZIAŁANIU MIKROKOMPUTERÓW

1.1. WPROWADZENIE	2
1.2. BUDOWA I DZIAŁANIE MIKROKOMPUTERÓW RODZINY MSC-51	5
1.2.1. ARCHITEKTURA MIKROKONTROLERA.	5
1.2.2. TRYBY PRACY MIKROKONTROLERA.	8
1.2.3. WEWNĘTRZNA PAMIĘĆ PROGRAMU I DANYCH.....	13
1.2.4. PORTY UNIWERSALNE P0..P3.	20
1.2.5. UKŁAD CZASOWO-LICZNIKOWY.	25
1.2.6. PORT TRANSMISJI SZEREGOWEJ.	40
1.2.7. KONTROLER PRZERWAŃ.	51
1.2.8. DODATKOWE URZĄDZENIA I/O MIKROKONTROLERA 89S8253.	57
1.3. PRACOWNIA SYSTEMÓW MIKROPROCESOROWYCH	63
1.3.1. SYSTEM DSM-51.	63
1.3.2. SYSTEM FTSM_51.....	65
1.3.3. ĆWICZENIA LABORATORYJNE.....	68
<u>ZADANIE 1</u> : GENERATOR ZDARZEŃ Z LICZNIKIEM T0.	68
<u>ZADANIE 2</u> : OBSŁUGA KLAWIATURY PROSTEJ I MULTIPLEKSOWANEJ.	69
<u>ZADANIE 3</u> : OBSŁUGA WYŚWIETLACZA MULTIPLEKSOWANEGO LED.....	72
<u>ZADANIE 4</u> : OBSŁUGA WYŚWIETLACZA LCD.	74
<u>ZADANIE 5</u> : OBSŁUGA PORTU SZEREGOWEGO (ŁĄCZA RS232).	81
<u>ZADANIE 6</u> : OBSŁUGA ŁĄCZA I ² C.	85
<u>ZADANIE 7</u> : OBSŁUGA ENKODERA OBROTOWEGO.	91
<u>ZADANIE 8</u> : OBSŁUGA SILNIKA KROKOWEGO.	93
<u>ZADANIE 9</u> : ZEGAR CZASU RZECZYWISTEGO.	96
<u>ZADANIE 10</u> : ZABEZPIECZANIE DZIAŁANIA SYSTEMU UKŁADEM WDT.	97

1.1. Wprowadzenie

O minikomputerach i mikrokontrolerach.

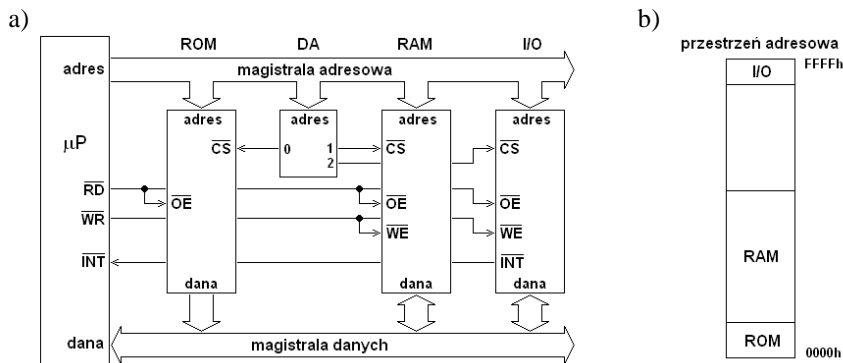
Banalną informacją jest to, że komputery całkowicie zdominowały rynek urządzeń, służących zarówno do produkcji dóbr jak i codziennego użytku. Praktycznie w każdym przypadku, czy jest to sterowanie obiektem przemysłowym czy też zarządzanie pracą telefonu komórkowego, schemat blokowy układu sterowania jest taki, jak pokazany na rys.1.1.1. Obiekt sterowania jest bezpośrednio kontrolowany przez system mikroprocesorowy, a sam system, przez człowieka (operatora). System mikroprocesorowy pozyskuje informacje o stanie obiektu sterowania za pośrednictwem zestawu urządzeń, które noszą wspólną nazwę urządzeń wejściowych (ang. input devices). Urządzenia te, przekształcają sygnały, np. analogowe, do postaci cyfrowej - dzięki temu będą mogły być one przetworzone przez układ mikroprocesorowy. Po przetworzeniu wprowadzonej informacji (zgodnie z oczekiwaniami operatora systemu) do obiektu sterowania przekazywana jest informacja o sposobie regulacji stanu obiektu. W tym przypadku, elementem pośredniczącym jest zestaw urządzeń, które noszą wspólną nazwę urządzeń wyjściowych (ang. output devices). Urządzenia te, przekształcają sygnały cyfrowe do postaci, np. analogowej - dzięki temu system mikroprocesorowy będzie mógł oddziaływać na analogowy obiekt sterowania.



Rys. 1.1.1. Schemat blokowy układu sterowania kontrolowanego przez system mikroprocesorowy [18].

W zdecydowanej większości przypadków życia codziennego, do sterowania pracą prostych obiektów użytkowych nie będą potrzebne systemy mikroprocesorowe o bardzo dużej mocy obliczeniowej - wystarczą systemy oparte o mikroprocesory 8-bitowe. Konstrukcja sprzętowa takich systemów nie jest skomplikowana. Przyglądając się różnym rozwiązaniom, można zauważyć, że typowy system mikroprocesorowy zawiera kilka, stale obecnych w systemie urządzeń - system zawiera mikroprocesor μP , pamięć programu ROM, pamięć danych RAM oraz zespół urządzeń wejścia/wyjścia, I/O (ang. input/output devices).

Różnice pomiędzy różnymi systemami są związane z urządzeniami I/O: z ich doбором i ilością.



Rys. 1.1.2. Schemat blokowy typowego systemu mikroprocesorowego.

Architekturę typowego minikomputera przedstawiono na rys.1.1.2. Wymiana informacji pomiędzy poszczególnymi urządzeniami systemu odbywa się liniami sygnałowymi, które są zgrupowane w tzw. magistrale: magistralę adresową, magistralę danych oraz magistralę sterującą. Jednostką sterującą pracą systemu jest mikroprocesor - to on decyduje o wymianie informacji w systemie. Z założenia, wszystkie urządzenia systemu są traktowane przez mikroprocesor jako zestaw rejestrów - każdy z rejestrów ma przydzielony w systemie unikalny adres. Ze względu na konstrukcję magistrali danych, mikroprocesor, w danym momencie, może dokonać wymiany informacji wyłącznie z jednym rejestrem systemu. Do wskazania adresu, w systemie wykorzystuje się układ pomocniczy - dekodery adresów, DA. Dzięki jego obecności, tzw. przestrzeń adresowa mikroprocesora jest podzielona na fragmenty, które przypisuje się poszczególnym urządzeniom systemu (rys.1.1.2b).

Podany wyżej opis konstrukcji systemu mikroprocesorowego, z założenia enigmatyczny, pozwala zauważyć, że w skład systemu wchodzi kilka (kilkanaście) specjalizowanych urządzeń, przystosowanych do współpracy z mikroprocesorem. Ponieważ technologia układów scalonych rozwija się bardzo dynamicznie, w chwili obecnej, proste systemy mikroprocesorowe są umieszczane w pojedynczych strukturach układu scalonego. Takie systemy są nazywane mikrokontrolerami i zawierają wszystkie elementy, pokazane na rys.1.1.2.

Kilka uwag o skrypcie.

Niniejszy skrypt napisano z myślą, że będzie on materiałem wystarczającym dla poznania struktury i sposobu działania mikrokontrolerów z rodziny MCS-51. Oczywiście, poznanie struktury i sposobu działania mikrokontrolera jest jedynie warunkiem wstępnym dla procesu jego programowania. Dlatego też, oprócz typowych rozważań na temat sprzętowej architektury mikrokontrolera, do skryptu dodano rozdział o sposobach programowania systemów komputerowych -

w części 2 skryptu omówiono kilka reguł poprawnego programowania systemów komputerowych. Przed przystąpieniem do programowania mikrokontrolerów zaleca się zapoznanie się z treścią tego rozdziału.

W skrypcie, dość często, występuje zamienne stosowanie słów *mikroprocesor* i *mikrokontroler*. Nie jest to błąd a jedynie chęć opisanie działania wybranego elementu komputera w przypadku, gdy działanie to jest powiązane z typowym oddziaływaniem jednostki centralnej CPU na środowisko systemu mikroprocesorowego.

W przypadku opisywania stanu bitów, pojęcie *ustawianie bitu* lub *bit ustawiony* może oznaczać zarówno stan zera lub jedynki logicznej tego bitu. Jeżeli w tekście nie zostanie to wyraźnie zaznaczone, należy traktować pojęcie *bit ustawiony* jako stan bitu o wartości 1 a pojęcie *ustawianie bitu*, jako zmianę stanu bitu z wartości 0 do 1.

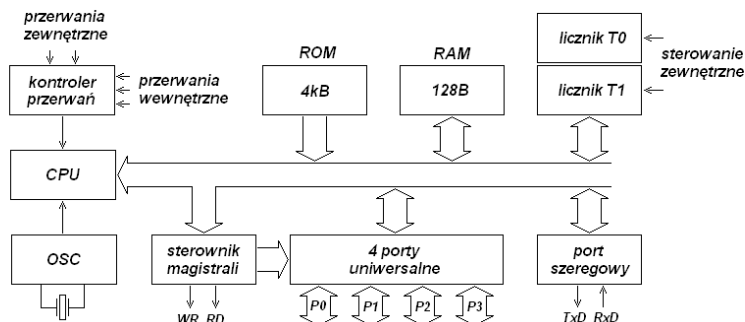
1.2. Budowa i działanie mikrokontrolerów rodziny MSC-51.

1.2.1. Architektura mikrokontrolera.

Jak już wspomniano, mikrokontrolery to minikomputery, które umieszczono w jednej strukturze układu scalonego. Zawierają one wszystkie niezbędne elementy typowego systemu mikroprocesorowego: CPU, pamięć ROM i RAM oraz zestaw kilku urządzeń I/O. Mikrokontrolery z rodziny MCS-51 zostały zaprojektowane w firmie Intel i pojawiły się na rynku pod koniec lat siedemdziesiątych ubiegłego wieku. Dobrze przemyślana konstrukcja i wygodny zestaw instrukcji spowodowały, że pomimo gwałtownego rozwoju rynku mikrokontrolerów, popularna od lat "pięćdziesiątka jedynka" ma się bardzo dobrze i jest w dalszym ciągu rozwijana.

Wszystkie mikrokontrolery należące do rodziny MCS-51 posiadają wspólny element, tzw. rdzeń mikrokontrolera, którego schemat blokowy pokazano na rys. 1.2.1. W skład rdzenia mikrokontrolera 80C51 (podstawowy przedstawiciel rodziny) wchodzi następujące elementy [1]:

- 8 bitowa jednostka centralna (mikroprocesor, CPU), która umożliwia przetwarzanie danych za pośrednictwem 111 rozkazów;
- wewnętrzna pamięć programu, ROM, która we współczesnych odpowiednikach jest prawie wyłącznie pamięcią typu EEPROM, pozwalającą na wielokrotne jej zapisywanie (np. do 10 000 razy w przypadku AT89S8253);
- wewnętrzna pamięć danych o pojemności 128 bajtów;
- układy czasowo-licznikowe T0 i T1;
- 4 uniwersalne, 8-bitowe porty I/O; P0..P3;
- port transmisji szeregowej;
- układ kontrolera przerw wewnętrznych i zewnętrznych;
- układ generatora sygnału zegarowego.



Rys. 1.2.1. Schemat blokowy struktury rdzenia 80C51 [1].

Większość współcześnie produkowanych mikrokontrolerów z rodziny MCS-51 posiada rdzeń układu 80C52, który był "następcą" 80C51. Różnica pomiędzy 80C51 a 80C52 jest związana z rozmiarem pamięci programu i danych oraz liczbą liczników układu czasowo-licznikowego (patrz tabela 1.2.1). Różnice pomiędzy innymi mikrokontrolerami, które należą do rodziny MCS-51, są związane z zestawem dodatkowych urządzeń I/O, które nie występują w rdzeniu mikrokontrolera 80C51/52. Ze względu na występowanie struktury podstawowej rdzenia we wszystkich mikrokontrolerach rodziny MCS-51, program napisany dla mikrokontrolera 80C51 może być uruchamiany praktycznie na każdym przedstawicielu rodziny.

Tabela 1.2.1. Porównanie parametrów kilku wybranych mikrokontrolerów z rodziny MCS-51.

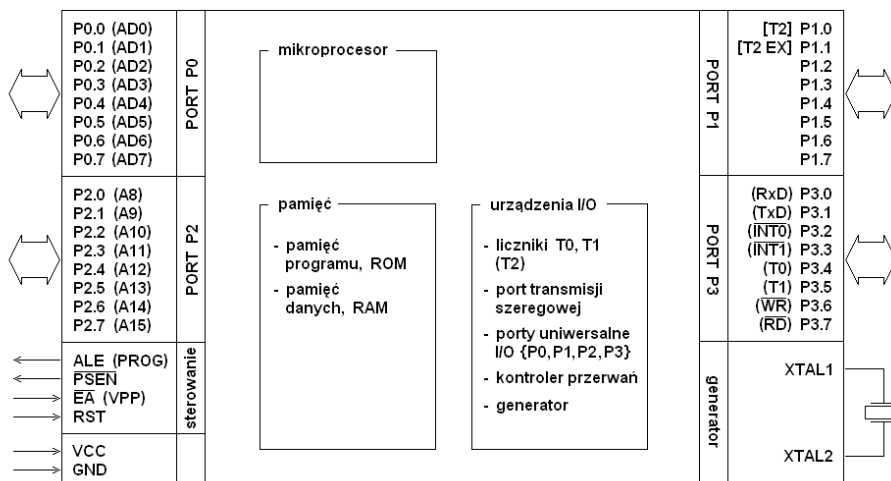
parametr	80C51	80C52	AT89S8253 ¹
rozmiar pamięci programu [kB]	4	8	8
rozmiar pamięci danych [B]	128	256	256
liczba układów czasowo-licznikowych	2	3	3
liczba źródeł przerwań	5	6	9
liczba wskaźników danych (DPTR)	1	1	2
łącze SPI	nie	nie	tak
pamięć EEPROM	nie	nie	tak
układ restartu awaryjnego (watchdog)	nie	nie	tak
programowanie mikrokontrolera po zamontowaniu na płycie (ISP)	nie	nie	tak

Mikrokontroler może wymieniać informację ze środowiskiem zewnętrznym za pośrednictwem 32 linii sygnałowych, standardowo przydzielonych do czterech, 8-bitowych portów, P0.. P3. Na rys.1.2.2, pokazano wszystkie wyprowadzenia (końcówki) mikrokontrolera 80C51/52. Dodatkowe cyfry, które występują w oznaczeniach linii sygnałowych portów P0..P3, wskazują na numer linii portu (numer bitu portu). Ze względu na ograniczoną liczbę wyprowadzeń, pozostałe urządzenia wewnętrzne I/O mają dostęp do zewnętrznego środowiska mikrokontrolera za pośrednictwem tych samych linii, które poprzednio przypisano portom P0..P3. Mówi się w takim przypadku o funkcji alternatywnej konkretnego wyprowadzenia. Na rysunku 1.2.2, funkcje alternatywne linii portów zaznaczono przez umieszczenie ich nazw w nawiasach.

Linie sygnałowe, które zgrupowano w bloku *sterowanie* są przeznaczone do zarządzania pracą mikrokontrolera i będą opisane szczegółowo w dalszej części opracowania. Skrócony opis wszystkich linii sygnałowych mikrokontrolera podano w tabeli 1.2.2.

¹ mikrokontroler systemu FTSM_51.

W opisie, pod pojęciem *port uniwersalny*, należy rozumieć to, że linie portu mogą pracować w trybie wejściowym, wyjściowym albo mieszanym.



Rys. 1.2.2. Końcówki mikrokontrolera 80C51/52.

Tabela 1.2.2. Skrócony opis końcówek mikrokontrolera 80C51/52.

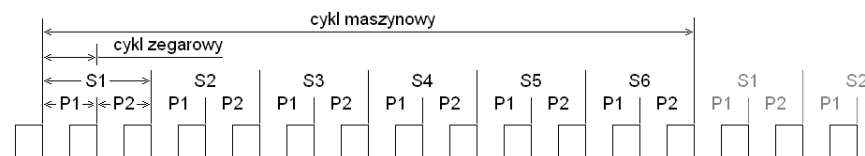
nazwa	opis przeznaczenia lub działania
VCC GND	Napięcie zasilania - najczęściej +5V (VCC) względem masy (GND);
PORT P0	Uniwersalny port 8 bitowy z tranzystorami typu "otwarty dren" - linie portu zdefiniowane jako wejściowe wymagają zewnętrznej polaryzacji. W mikroprocesorowym trybie pracy, port przynosi informację o młodszej części adresu oraz danej.
PORT P1	Uniwersalny port 8 bitowy z rezystorowym układem wewnętrznej polaryzacji. Linie P1.0 i P1.1 mogą alternatywnie wspierać działanie licznika T2 (w 80C52).
PORT P2	Uniwersalny port 8 bitowy z rezystorowym układem wewnętrznej polaryzacji. W mikroprocesorowym trybie pracy, linie portu przenoszą informację o starszej części adresu.
PORT P3	Uniwersalny port 8 bitowy z rezystorowym układem wewnętrznej polaryzacji. Wszystkim liniom przypisane są funkcje alternatywne.
RST	Reset - wejście kasowania mikrokontrolera.
ALE	Address Latch Enable - wyjście sygnału sterującego, umożliwiającego przechwycenie informacji o młodszej części adresu z linii portu P0. Sygnał jest wykorzystywany w trybie mikroprocesorowym pracy mikrokontrolera.

PSEN	Program Store Enable - wyjście sygnału sterującego, będącego odpowiednikiem sygnału RD i przeznaczonego do odczytu kodu programu z zewnętrznej pamięci programu. Sygnał jest używany w trybie mikroprocesorowym pracy mikrokontrolera .
EA	External Access Enable - wejście sygnału sterującego, wskazującego lokalizację pamięci programu. Gdy EA=0 to kod programu jest pobierany z zewnętrznej pamięci programu; gdy EA=1 to program jest ulokowany wewnątrz struktury mikrokontrolera.
XTAL1 XTAL2	Miejsce dołączenia oscylatora kwarcowego.

1.2.2. Tryby pracy mikrokontrolera.

Cykl zegarowy, maszynowy i rozkazowy.

Wszystkie działania wewnątrz struktury mikrokontrolera są synchronizowane przez periodyczny sygnał, wytworzony w generatorze mikrokontrolera. Sygnał ten jest nazywany sygnałem zegarowym albo zegarem systemowym - w dalszej części opracowania będzie oznaczany symbolem: **OSC** w przypadku podkreślenia źródła pochodzenia sygnału lub **f_{osc}** w przypadku chęci zaznaczenia częstości drgań tego sygnału. Praca generatora jest stabilizowana przez zewnętrzny rezonator kwarcowy, który jest dołączany do końcówek, XTAL1 oraz XTAL2 (rys.1.2.2., tabela 1.2.2). Zastosowanie kwarcu zapewnia wygenerowanie sygnału zegarowego o dużej stabilności długoterminowej. Przeciętna stabilność częstości rezonatora kwarcowego wynosi ok. 20-50ppm. Okres sygnału zegarowego mikrokontrolera jest wyznaczany przez częstotliwość rezonansową kwarcu. Każdy takt sygnału zegarowego, wyznaczany opadającym zboczem sygnału, jest nazywany **cyklem zegarowym**. Podstawowe działania typowego mikroprocesora są wykonywane w ramach kilku lub kilkunastu taktów sygnału zegarowego. Takie działania noszą nazwę **cyklu maszynowego**. Pełne wykonanie rozkazu wiąże się z wykonaniem jednego lub kilku cykli maszynowych a grupa tych cykli nosi nazwę **cyklu rozkazowego**. W mikrokontrolerach z rodziny MCS-51 przyjęto, że każdy cykl maszynowy jest wykonywany w ramach 12 cykli zegarowych. Przy założeniu, że sygnał zegarowy ma częstotliwość 12MHz, czas trwania cyklu maszynowego wynosi 1μs.



Rys. 1.2.3. Cykl maszynowy i jego fazy.

Na rys.1.2.3 pokazano cykl zegarowy i maszynowy mikrokontrolera 80C51. Dla wygody opisu przekazywania informacji w strukturze mikrokontrolera, poszczególne takty sygnału zegarowego zostały oznakowane. Cykl maszynowy podzielono na 6 stanów (S1..S6), a do każdego z nich przypisano po 2 cykle zegarowe (P1 i P2). Należy w tym miejscu zauważyć, że najnowsze generacje mikrokontrolerów rodziny MCS-51 mogą pracować w trybie, w którym cykl maszynowy jest wykonywany przez mniejszą liczbą cykli zegarowych [12].

Wymiana informacji z urządzeniami zewnętrznymi.

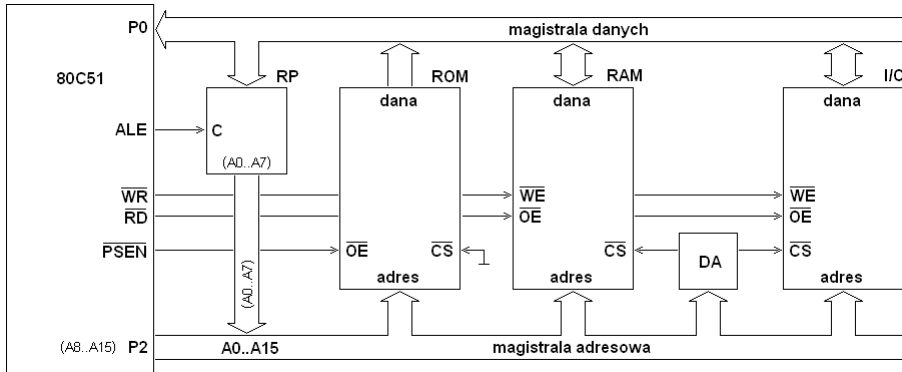
Mikrokontrolery 80C51/52 posiadają wewnętrzną pamięć ROM i niewielką, wewnętrzną pamięć RAM. Rozmiar pamięci może być powiększony przez zastosowanie zewnętrznych układów pamięciowych. Mikrokontrolery są w stanie adresować zewnętrzne pamięci ROM i RAM o rozmiarze do 64kB każda. Ze względu na konieczność stworzenia w takim przypadku zewnętrznej magistrali adresowej, danych i sterowania, trzeba zrezygnować z części linii sygnałowych, do tej pory używanych jako uniwersalne linie I/O. Taki sposób pracy mikrokontrolera będzie nazywany **trybem mikroprocesorowym**.



tryb mikroprocesorowy mikrokontrolera to taki rodzaj jego pracy, w którym część linii I/O przekształca się w magistrale mikroprocesora (adresową, danych i sterowania) ..

Dla zdecydowanej większości przypadków, rozmiar pamięci wewnętrznej jest wystarczający do wykorzystania mikrokontrolera zgodnie z jego przeznaczeniem. Ten tryb pracy nie nosi żadnej szczególnej nazwy. Pojawienie się pojęcia trybu mikroprocesorowego oznacza jedynie potrzebę rezygnacji z 18 linii portów uniwersalnych na rzecz magistrali mikroprocesorowej oraz obecność w systemie mikroprocesorowym dodatkowej pamięci lub urządzeń I/O.

Przekazywanie danej w typowym systemie mikroprocesorowym odbywa się za pośrednictwem dwukierunkowej magistrali danych. Informacja o adresie i sygnały sterowania zapisem lub odczytem są przesyłane magistralami jednokierunkowymi. W mikrokontrolerze 80C51, do przesyłania danej w trybie dwukierunkowym, przystosowano port P0 (struktura sprzętowa portu zastała omówiona w rozdziale 1.1.1). Informacja o starszej części adresu jest przekazywana za pośrednictwem portu P2. Młodsza część adresu jest przekazywana przez port P0 w momencie, gdy sygnał ALE jest w stanie aktywnym - jest jedyką logiczną. Informacja o młodszej części adresu musi być przechwycona i zapamiętana przez dodatkowy układ scalony - 8-bitowy rejestr zatrząskowy. Taki sposób przekazywania informacji o adresie powoduje, że system mikroprocesorowy, oparty o mikrokontroler rodziny MCS-51 powinien wyglądać tak, jak to pokazano na rys.1.2.4.



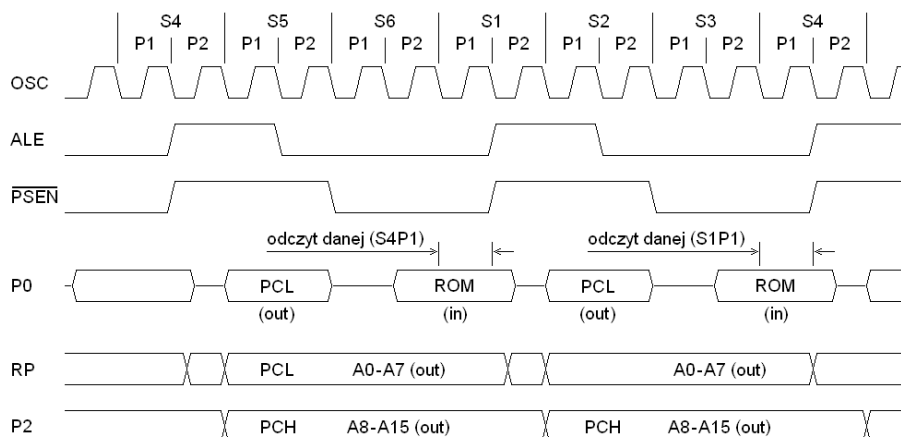
Rys. 1.2.4. Architektura systemu mikroprocesorowego z zewnętrzną pamięcią programu i danych oraz zewnętrznym urządzeniem I/O.

Na pokazanym wyżej rysunku, młodsza część adresu jest przechwytywana przez układ RP. Kod rozkazu, wraz z operandami, jest umieszczony w pamięci ROM. Odczytywanie elementów rozkazu odbywa się w momencie, gdy adres jest ustabilizowany a linia PSEN przechodzi w stan aktywny - w stan zera logicznego. Odczytywanie lub zapisywanie danej do pamięci RAM odbywa się w momencie, gdy adres jest ustabilizowany a linia RD lub WR przechodzi w stan aktywny - w stan zera logicznego. Odczytywanie danej jest związane z aktywnym stanem linii RD a zapisywanie danej - z aktywnym stanem linii WR. W identyczny sposób, jak z pamięcią RAM, jest realizowana wymiana informacji z urządzeniami I/O. W mikrokontrolerach rodziny MCS-51 nie występuje formalny podział na urządzenia I/O oraz pamięć RAM. O tym, czy mikrokontroler wymienia informację z zewnętrzną pamięcią danych czy też z zewnętrznym urządzeniem I/O, decyduje wyłącznie adres przypisany urządzeniu. W takim przypadku, aktywacja wskazanych elementów otoczenia odbywa się za pośrednictwem zewnętrznego dekodera adresów - DA.

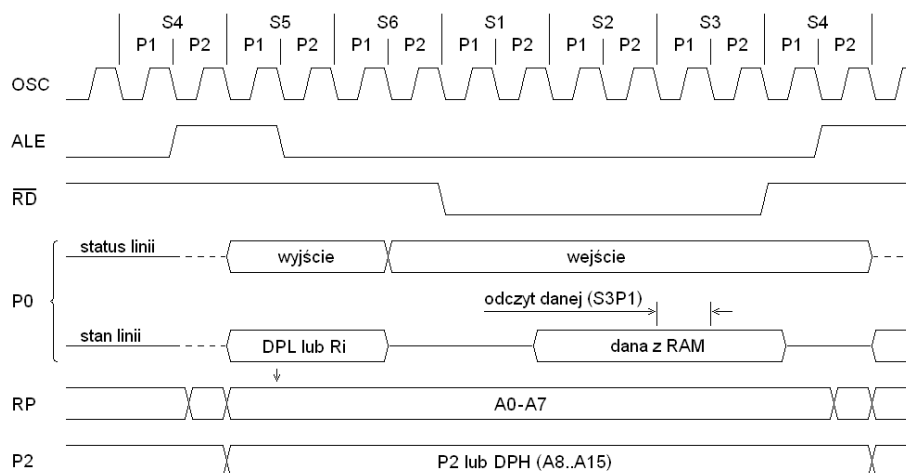
Na rysunku 1.2.5 pokazano przebiegi sygnałów w trakcie wykonywania cyklu maszynowego pobierania elementów kodu rozkazu. W tym przypadku, o adresie decyduje stan licznika rozkazów - PC. Młodsza część licznika, PCL, jest przekazywana do portu P0 w momencie, gdy sygnał ALE jest jedynką logiczną. Starsza część licznika, PCH, jest przekazywana do portu P2. Odczytywanie informacji z zewnętrznej pamięci danych następuje w ostatniej fazie aktywności sygnału PSEN (S4P1 w przypadku odczytywania kodu; S1P1 w przypadku odczytywania pierwszego operandu - drugi operand, jeżeli istnieje, jest odczytywany w fazie S4P1 następnego cyklu maszynowego).

Wymiana informacji, np. z zewnętrzną pamięcią RAM może się odbywać w dwu trybach adresowania: w trybie 8-bitowym i 16-bitowym. Do adresowania 8-bitowego wykorzystuje się rejestr R0 lub R1. Do adresowania 16-bitowego używany jest rejestr DPTR, który złożony jest z dwóch rejestrów 8-bitowych: z rejestru DPL i DPH. Młodsza część adresu jest przechowywana w rejestrze DPL, a starsza w DPH.

W obu przypadkach, do wymiany informacji z zewnętrzną pamięcią RAM (I/O) jest używana instrukcja MOVX (patrz lista instrukcji, str.198). Różnica w sposobach adresowania sprowadza się do tego, że adresowanie 8-bitowe nie zmienia stanu portu P2. Przy korzystaniu z małej pamięci RAM, ograniczonej do 256 adresów, port P2 może być wykorzystywany do innych celów niż adresowanie. W przypadku jednak, gdy oprócz zewnętrznej, małej pamięci RAM, w systemie jest używana zewnętrzna pamięć programu, to port P2 nie nadaje się do innych czynności niż adresowanie.



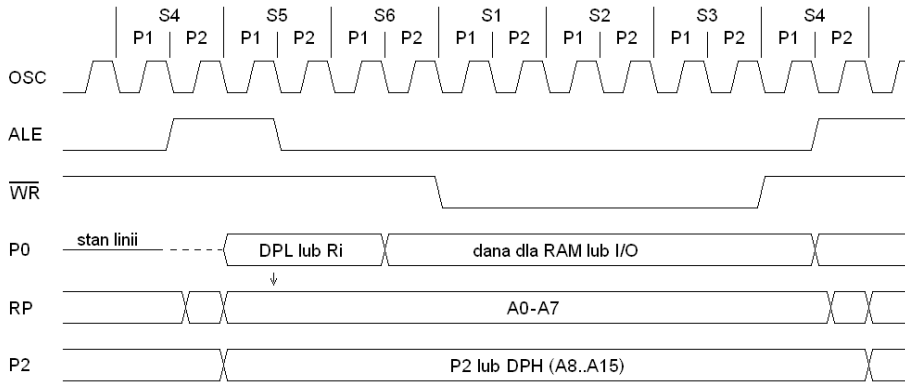
Rys. 1.2.5. Cykl maszynowy odczytu danych z zewnętrznej pamięci programu.



Rys. 1.2.6. Cykl maszynowy odczytu danych z zewnętrznej pamięci RAM lub urządzenia I/O.

Na rysunkach 1.2.6 i 1.2.7 pokazano przebiegi sygnałów w trakcie wykonywania cyklu maszynowego, odpowiednio: odczytywania danych z pamięci zewnętrznej RAM (lub I/O) i zapisywania danych do pamięci zewnętrznej RAM (lub I/O). W przypadku odczytywania danych, pojawienie się stanu aktywnego linii

RD powinno spowodować wprowadzenie danej z zewnętrznej pamięci RAM na linie portu P0. Odczytywanie stanu portu następuje w fazie S3P1. W przypadku zapisywania danej, dana ta jest wprowadzana na linie portu P0 przed wygenerowaniem stanu aktywnego linii WR. Stan aktywny linii WR jest informacją dla zewnętrznej pamięci RAM o gotowości danej do zapisu. W obu przypadkach, odczytu lub zapisu danej, stan portu P2 jest zmieniany w przypadku adresowania 16-bitowego - do portu jest wprowadzany stan rejestru DPH.



Rys. 1.2.7. Cykl maszynowy zapisu danej do zewnętrznej pamięci RAM lub urządzenia I/O.

Praca w trybie energooszczędnym.

Mikrokontrolery rodziny MCS-51 posiadają opcje pracy w trybie energooszczędnym. Jest to bardzo ważna właściwość w przypadkach, gdy urządzenie mikroprocesorowe jest zasilane z baterii ogniwo, a właściwe działanie urządzenia jest wykonywane co pewien czas. Mikrokontrolery 80C51/52 są wyposażone w 2 tryby zmniejszonego poboru energii: tryb pracy jałowej (ang. idle mode) oraz tryb obniżonego poboru mocy (ang. power down mode). Mikrokontroler można przeprowadzić w jeden ze wskazanych trybów przez ustawienie bitu PD lub IDL - bity są umieszczone w rejestrze PCON (patrz opis rejestru na str.51).

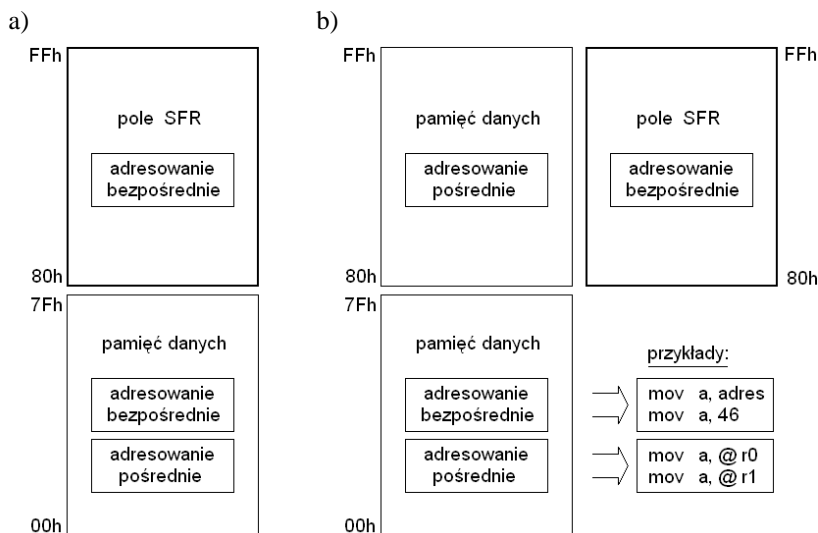
W trybie jałowym pracują wszystkie urządzenia I/O mikrokontrolera za wyjątkiem procesora - wykonywanie programu jest zawieszona. Wyjście z trybu pracy jałowej jest możliwe przez skasowanie mikrokontrolera lub przez zgłoszenie dowolnego przerwania przez aktywny system przerwania. Po zgłoszeniu przerwania, bit IDL jest kasowany sprzętowo. Tryb pracy jałowej pozwala na kilkukrotne zmniejszenie prądu zasilania w stosunku do trybu pracy normalnej.

W trybie pracy z obniżonym poborem mocy, wyłączane są wszystkie układy mikrokontrolera ale nie jest odcinane zasilanie wewnętrznej pamięci RAM - pamięć nie traci informacji. Wyjście z trybu obniżonego poboru mocy jest możliwe przez kasowanie mikrokontrolera lub przez zgłoszenie zewnętrznego przerwania liniami INT0 lub INT1. Po zgłoszeniu przerwania, bit PD jest kasowany sprzętowo. Tryb pracy z obniżonym poborem mocy pozwala na kilkusetkrotne zmniejszenie prądu zasilania w stosunku do trybu pracy normalnej.

1.2.3. Wewnętrzna pamięć programu i danych.

Pole pamięci danych - RAM.

Strukturę wewnętrznej pamięci RAM przedstawiono na rysunku 1.2.8. Można ją podzielić na 2 podstawowe bloki: pole pamięci danych oraz pole tzw. rejestrów specjalnych, SFR (ang. Special Function Registers).



Rys. 1.2.8. Struktura wewnętrznej pamięci RAM dla 80C51 (a) i 80C52 (b).

Rozmiar pola pamięci danych, w układzie 80C51 wynosi 128 bajtów (rys. 1.2.8a) a w układzie 80C52, 256 bajtów (2*128 bajtów, (rys.1.2.8b)). Dla wygody opisu, pole pamięci o adresach zawartych w przedziale 0..127, często nazywa się *pamięcią dolnego obszaru RAM* lub krócej, *pamięcią dolną*. Analogicznie, pole pamięci z adresami zawartymi w przedziale 128..255 nazywane jest *pamięcią górnego obszaru RAM* lub *pamięcią górną*. W obu mikrokontrolerach pole pamięci danych rozpoczyna się od adresu 0 i jest to pole ciągłe - do każdego adresu jest przydzielony 8-bitowy rejestr danej. Dla pola SFR przydzielono adresy z zakresu 128..255. W odróżnieniu od pola danych, pole SFR nie jest obsadzone przez rejestry w sposób ciągły - jest w nim sporo luk (patrz tabela 1.2.4).

Przyglądając się rysunkowi 1.2.8b. opisującego strukturę pamięci w układzie 80C52, można zauważyć, że zarówno polu SFR jak i górnej pamięci danych przypisany jest ten sam zestaw adresów. Rozróżnienie tych obszarów jest realizowane przez sposób zapisywania i odczytywania danej. Pierwszym sposobem jest odczytywanie i zapisywanie danej instrukcjami, do wykonania których jest niezbędne podanie wartości adresu rejestru. Mówi się w takim przypadku o instrukcjach dostępu bezpośredniego - o adresowaniu bezpośrednim.

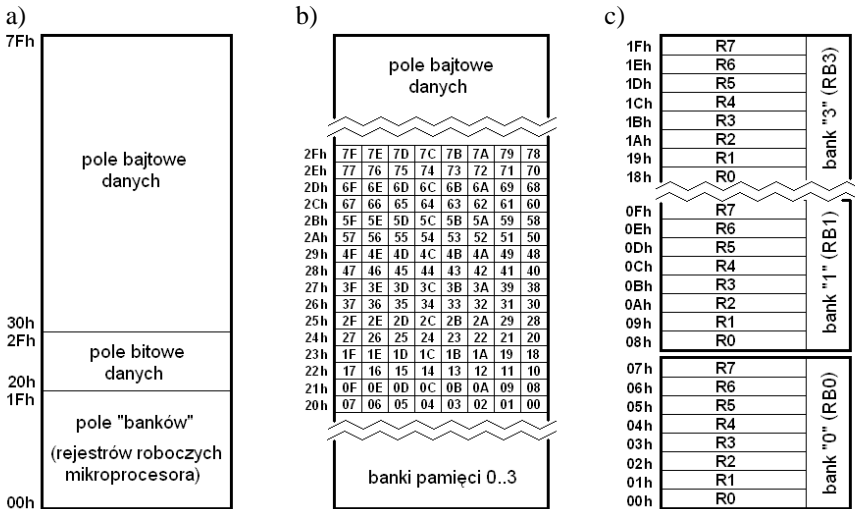
Drugim sposobem jest tzw. adresowanie pośrednie, które jest wykonywane za pośrednictwem rejestrów o nazwach R0 i R1. Przed zapisem lub odczytem danej, do rejestrów tych wprowadza się wartość adresu docelowego a następnie wykonuje operację odczytu lub zapisu przystosowaną do tego celu instrukcją. Dostęp do pola SFR jest realizowany poprzez adresowanie bezpośrednie, a dostęp do pola górnej pamięci RAM - przez adresowanie pośrednie. W polu pamięci dolnej, dopuszczalne jest adresowanie obydwoma sposobami.



adresowanie bezpośrednie i pośrednie to sposób wskazania adresu w polu wewnętrznej pamięci RAM - każdemu ze sposobów jest przydzielony oddzielny zestaw instrukcji ..

Na rys.1.2.9a, pokazano strukturę dolnej części pamięci RAM. Ze względu na sposób używania oraz sposób zapisywania i odczytywania danych, pamięć tego obszaru można podzielić na 3 grupy:

- pole 4 banków rejestrów roboczych mikroprocesora;
- pole dla danych jednobitowych;
- pole dla danych wyłącznie 8-bitowych.



Rys. 1.2.9. Struktura dolnej części pamięci RAM: podział formalny pola (a), pole bitowe (b) i pole banków (c).

Pole danych 8-bitowych, rozpoczynające się od adresu 30h (48), jest klasycznym polem pamięci RAM, które umożliwia zapisywanie i odczytywanie słów 8-bitowych - bajtów. Taki sam sposób zapisywania i odczytywania informacji jest stosowany dla górnego pola pamięci RAM, o adresach 128..255.

Pole bitowe jest przeznaczone do zapamiętywania informacji bitowej. Do tego celu zarezerwowano 128 bitów pamięci, zgrupowanych w 16 bajtach. Pole

rozpoczyna się od adresu 20h (32) i kończy adresem 2Fh (47). Każdy bit tego obszaru ma przydzielony indywidualny numer (adres) z zakresu od 00h do 7Fh (0..127). Wykaz numerów indywidualnych poszczególnych bitów pola bitowego jest pokazany na rys.1.2.9b. Wpisanie lub odczytanie stanu bitu jest wykonywane przez instrukcje operacji bitowych, w których numer bitu jest używany w sposób bezpośredni. Oczywiście, w obszarze pola bitowego dozwolone jest również zapamiętywanie bajtów. Zapisywanie lub odczytywanie bajtów odbywa się za pośrednictwem instrukcji adresowania bezpośredniego lub pośredniego.

Trzecie z pól, które wskazano na rys.1.2.9c, nazwano polem banków rejestrów roboczych mikroprocesora. Mikroprocesor układu 80C51/52, jak każdy inny, potrzebuje do wykonywania rozkazów grupy rejestrów roboczych, z których część nazywa się rejestrami ogólnego przeznaczenia (ang. GPR, General Purpose Registers). Grupa ta jest ulokowana w obszarze pamięci RAM i obejmuje sobą 8 rejestrów. Rejestry są numerowane od wartości 0 do 7 i noszą kolejne oznaczenia: R0, R1, R2 ..R7. Dwa spośród nich, R0 i R1, są szczególnie wyróżnione bo oprócz standardowego sposobu ich używania, wykorzystywane są do adresowania pośredniego. Zestaw 8 rejestrów, R0..R7, nazywany jest bankiem rejestrów lub krócej bankiem.

Jak widać z rys.1.2.9c, w pamięci RAM umieszczono 4 banki rejestrów roboczych, RB0..RB3 (ang. Registers Bank). Mikroprocesor do bieżącej pracy wykorzystuje tylko jeden bank. Przełączanie banków odbywa się poprzez odpowiednie ustawienie dwu bitów, RS0 i RS1, które umieszczono w słowie stanu programu, PSW. Po włączeniu zasilania i wstępnym wykasowaniu mikrokontrolera, do współpracy z mikrokontrolerem jest wyznaczany bank RB0. Ponieważ duża grupa instrukcji mikrokontrolera jest wykonywana nie poprzez odwołanie się do bezwzględnego adresu pamięci ale do adresu przypisanego konkretnemu rejestrowi roboczemu aktywnego banku, sam fakt wyboru innego banku powoduje automatyczną zmianę aż 8 adresów jednocześnie. Przykładowo, w zależności od numeru wybranego banku, rejestr R0 może wskazywać na adresy 0, 8, 16 i 24; rejestr R1 na adresy 1, 9, 17 i 25, itd.

Błyskawiczna wymiana adresów jest bardzo użyteczna w przypadku obsługi przerwania. W "klasycznej" metodzie obsługi przerwania, w jego fazie początkowej trzeba przenieść na stos zawartość wszystkich rejestrów, które będą używane do wykonania programu obsługi przerwania. W fazie końcowej występują czynności odwrotne - ze stosu pobiera się dane o stanie rejestrów i przywraca ich pierwotny stan. Program obsługi przerwania powinien być realizowany w czasie najkrótszym z możliwych (patrz uwagi w rozdziale 1.4.3, str.117) a wszystkie czynności zapisu na stos i odczytu ze stosu zabierają sporo czasu. Możliwość przełączenia banku, np. z banku RB0 na RB1 powoduje, że od momentu przełączenia, program przerwania jest wykonywany na wartościach rejestrów R0..R7 z banku RB1, a dane w banku RB0 pozostają nienaruszone - są elementami swojego rodzaju stosu. Powrót do banku RB0 powoduje natychmiastowe odzyskanie stanu aż 8 rejestrów.

Drugim blokiem struktury wewnętrznej pamięci RAM jest pole rejestrów specjalnych, SFR. "Specjalność" rejestrów z grupy SFR polega na tym, że w odróżnieniu od typowej pamięci RAM, część z nich posiada połączenie fizyczne z urządzeniami I/O i może to być traktowane przez mikroprocesor w dwojaki sposób: jako rejestry typowej pamięci RAM albo jako urządzenia I/O (a dokładniej, jako rejestry pamięciowe urządzeń I/O). Jakikolwiek zapis danej do takiego rejestru, powoduje możliwość pojawienia się tej informacji na wyprowadzeniach zewnętrznych mikrokontrolera. Jakakolwiek zmiana stanu wyprowadzeń mikrokontrolera, wykonana przez zewnętrzne urządzenia I/O, może być bezpośrednio odczytana przez CPU ale nie naruszana jest w takim przypadku informacja zapisana w rejestrze. Mikroprocesor może zatem wykonywać na takim rejestrze pewną grupę instrukcji i oddziaływać na stan wyjść wynikiem działania takiej instrukcji. Grupa instrukcji, która może być wykonywana na rejestrach SFR, nazywana jest instrukcjami RMW (ang. Read-Modify-Write Instructions). Wykaz tych instrukcji podano w tabeli 1.2.3.

Pozostałe rejestry obszaru SFR, które nie mające sprzężenia ze sprzętem, są traktowane przez CPU jako grupa rejestrów własnych mikroprocesora: akumulator ACC, akumulator pomocniczy B, rejestr stanu programu PSW, wskaźnik stosu SP, rejestry adresowania pośredniego DPL i DPH (DPTR). Rejestr licznika rozkazów, PC, jest umiejscowiony bezpośrednio w strukturze mikroprocesora. Wykaz rejestrów pola SFR mikrokontrolera 80C52 podano w tabeli 1.2.4 (rejestry mikrokontrolera AT89S8253 pokazano w dodatku 1.3.5). Rejestry pozwalające na operacje bitowe umieszczone są w pierwszej kolumnie tabeli.

Tabela 1.2.3. Wykaz instrukcji RMW.

instrukcja	przykład	opis
ANL	ANL P1,A	iloczyn logiczny
ORL	ORL P2,A	suma logiczna
XRL	XRL P3,A	suma modulo 2
JBC	JBC P1,1, etykieta	skok gdy bit=1 i kasowanie bitu
CPL	CPL P3.0	negacja bitu
INC	INC P2	zwiększanie stanu o 1
DEC	DEC P3	zmniejszanie stanu o 1
DJNZ	DJNZ P3, etykieta	zmniejszanie stanu o 1 i skok gdy zero
MOV Px.y,C	MOV P1.4,C	kopiowanie bitu CY do bitu y w porcie x
CLR Px.y	CLR P2.3	kasowanie bitu y w porcie x
SETB Px.y	SETB P3.1	ustawianie bitu y w porcie x

Tabela 1.2.4. Pole SFR mikrokontrolera 80C52

0F8h								0FFh
0F0h	B 00000000							0F7h
0E8h								0EFh
0E0h	ACC 00000000							0E7h
0D8h								0DFh
0D0h	PSW 00000000							0D7h
0C8h	T2CON 00000000	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFh
0C0h								0C7h
0B8h	IP xx000000							0BFh
0B0h	P3 11111111							0B7h
0A8h	IE 0x000000							0AFh
0A0h	P2 11111111							0A7h
098h	SCON 00000000	SBUF xxxxxxx						09Fh
090h	P1 11111111							097h
088h	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		08Fh
080h	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000			PCON 0xxx0000	087h

Stos.

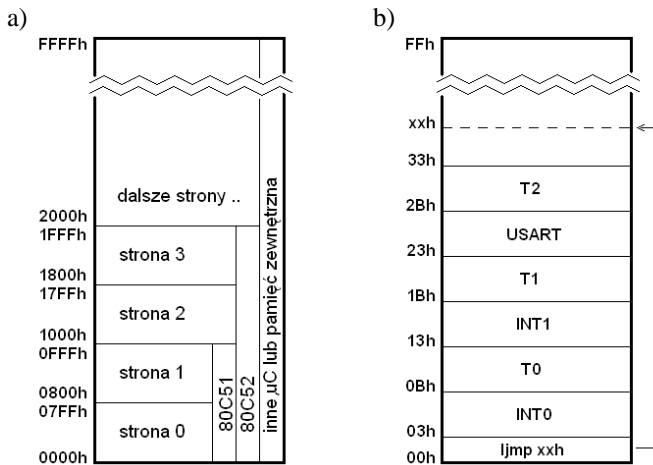
Stos jest wydzielonym polem pamięci RAM, do którego stosuje się specyficzny sposób zapisywania i odczytywania informacji - zmianę stanu stosu uzyskuje się poprzez operacje wykonywane wyłącznie na tzw. wierzchołku stosu. (tylko wierzchołek stosu jest widoczny dla mikroprocesora). Pobieranie jakiegokolwiek danej ze stosu dotyczy wyłącznie danej składowanej tam (zapisanej) jako ostatnia. Informacja o adresie wierzchołka stosu jest przechowywana w rejestrze nazywanym wskaźnikiem stosu, SP (ang. Stack Pointer). W mikrokontrolerach rodziny MCS-51, przed każdorazowym zapisem danej na stos, wskaźnik stosu jest zwiększany o 1 a po odczycie, zmniejszany o 1.

Po uruchomieniu mikrokontrolera 80C51/52, do rejestru SP wpisywana jest automatycznie wartość 07h co jest równoważne temu, że pierwszy bajt stosu może być zapisany pod adres 08h (SP+1). Stos można ulokować w dowolnym miejscu pamięci RAM poprzez wpisanie do rejestru SP właściwej wartości. Przy doborze lokalizacji stosu należy pamiętać o fakcie, że wskaźnik stosu "rośnie". Nie może dojść do sytuacji, w której dane stosu zmieniają inne dane albo zostaną przekroczone granice pola RAM. W czasie programowania mikrokontrolera, przed ustawieniem adresu początkowego stosu albo położenia pola danych, istnieje potrzeba oszacowania maksymalnego rozmiaru tego stosu.

W mikrokontrolerach rodziny MCS-51, stos zawsze jest umieszczony w obszarze wewnętrznej pamięci RAM i nie jest możliwe jego przeniesienie do zewnętrznej pamięci RAM (istnieją wyjątki, gdy mikrokontroler ma wbudowaną, dodatkową pamięć, nazywaną pamięcią XRAM [12]).

Pole pamięci programu - ROM.

Strukturę wewnętrznej pamięci programu przedstawiono na rys.1.2.10. Pomimo ciągłości obszaru pamięci, na rysunku 1.2.10a można zauważyć formalny podział na tzw. strony. Każda ze stron ma rozmiar 2kB. Istnienie podziału na strony jest determinowane istnieniem 2 rozkazów, *ajmp* i *acall*, które posługują się adresem 11-bitowym i tym samym ograniczają zasięg stosowalności do pola o rozmiarze pojedynczej strony. Pojęcie strony staje się nieistotne w przypadku posługiwania się instrukcjami *ljmp* i *lcall*, które operują adresem 16-bitowym.



Rys. 1.2.10. Struktura pamięci programu: podział formalny pola pamięci (a) i mapa pola pierwszych 256 bajtów (b).

Na rys.1.2.10b, pokazano fragment początkowy pola pamięci programu. Od adresu 03h (3), z krokiem co 8, umieszczone są grupy bajtów pamięci, które przypisano programom obsługi przerwań. Każdemu z przerwań przypisano adres początkowy programu obsługi przerwania - często ten adres nazywany jest *wektorem przerwania*. Położenie wektorów jest niezmiennie i determinuje "ważność przerwania" - priorytet przerwania. Zdecydowano, że priorytet będzie najwyższy dla wektora o najniższym adresie i będzie malał wraz ze wzrostem wartości adresu tego wektora. Priorytet przerwania można częściowo zmienić poprzez uaktywnienie wyższego poziomu sterowania przerwaniem (patrz rozdział 1.2.7). Przyjęto, że na poziomie podstawowym sterowania, przerwanie od urządzeń I/O będą przyjmowane wg następującej kolejności:

03h - od urządzenia zewnętrznego, z wejścia INT0;

0Bh - od licznika T0;

- 13h** - od urządzenia zewnętrznego, z wejścia INT1;
- 1Bh** - od licznika T2;
- 23h** - od portu transmisji szeregowej;
- 2Bh** - od licznika T2 (dla 80C52)

Po wykasowaniu mikrokontrolera stanem aktywnym sygnału RST, program jest wykonywany od adresu 0. W przypadku zrezygnowania z usług przerwanych, program może zajmować kolejne adresy pamięci ROM, rozpoczynając od adresu 0 i wszystkie uczynione do tej pory uwagi o rezerwacji pamięci są nieistotne. W przypadku jednak, gdy wymagana jest obsługa nawet jednego przerwania, program musi "ominać" procedurę obsługi tego przerwania. Można to osiągnąć przez wykonanie instrukcji skoku do dalszej części programu (patrz rys.1.2.10b). Zajmującą najwięcej pamięci jest instrukcja skoku dalekiego *ljmp* - do jej wykonania potrzebne są 3 bajty pamięci. Z tego właśnie powodu procedury obsługi przerwania rozpoczynają się od adresu 3 a nie od adresu 0 - adresy 0..2 zarezerwowano na instrukcję skoku.

Przyglądając się polu pamięci programu, można się zdziwić, że poszczególnym przerwaniom przypisano zaledwie 8 bajtów pamięci. W większości przypadków ta wielkość jest wystarczająca dla poprawnie skonstruowanego programu obsługi przerwania (patrz uwagi w rozdziale 1.4.3, str.117). Gdy wymagany jest większy obszar pamięci, można ją "rozszerzyć" instrukcją skoku (np. *ljmp*) lub wywołania (np. *lcall*).

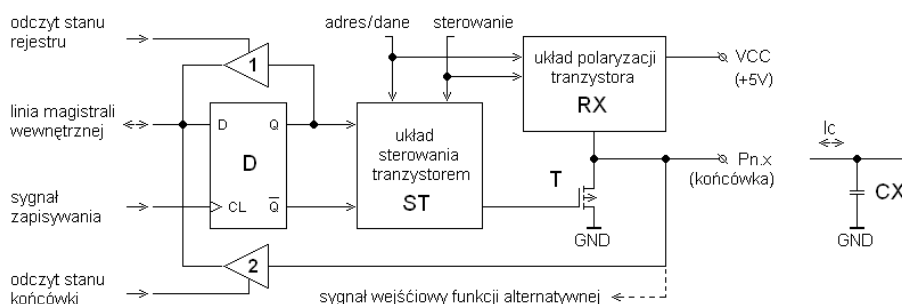
Podane wyżej informacje przypisano organizacji i działaniu wewnętrznej pamięci programu. Identyczny schemat obowiązuje w przypadku stosowania zewnętrznej pamięci programu. Mikrokontrolery z rodziny MCS-51 mogą się posługiwać zarówno pamięcią wewnętrzną programu jak i pamięcią zewnętrzną. Wybór pamięci wewnętrznej jest determinowany przez podanie jedynek logicznych na wejście sterujące EA (External Access). W przypadku przekroczenia przez program pojemności pamięci wewnętrznej, np. po wykonaniu instrukcji skoku, mikrokontroler, w sposób automatyczny rozpocznie pobieranie kodu programu z zewnętrznej pamięci ROM. Pomimo istnienia możliwości pracy z wewnętrzną i zewnętrzną pamięcią programu, takie rozwiązanie jest mało praktyczne. Lepszym rozwiązaniem jest rezygnacja z pamięci wewnętrznej ROM i wykorzystanie wyłącznie pamięci zewnętrznej. Można to uczynić przez dołączenie końcówki EA do masy elektrycznej (podanie zera logicznego). W takim przypadku, program jest wykonywany wyłącznie z pamięci zewnętrznej.

I jeszcze jedna uwaga. W pamięci ROM, oprócz kodu programu, można również przechowywać dane. Wśród instrukcji mikrokontrolera istnieje jedna, przeznaczona do odczytywania danej z pamięci programu i nosi ona oznaczenie *movc*. Opis działania tej instrukcji podano na str.196.

1.2.4. Porty uniwersalne P0..P3.

Jak już wspomniano, do wymiany informacji z otoczeniem, mikrokontrolery używają linii sygnałowych pogrupowanych w tzw. porty. Na rysunku 1.2.11 pokazano ogólny schemat struktury pojedynczej linii portu a na rysunku 1.2.12 strukturę poszczególnych portów, P0..P3.

Do każdego wyprowadzenia (końcówki) portów P0..P3 przypisano tranzystor wyjściowy T, przerzutnik typu D oraz 2 bramki trójstanowe, 1 i 2. Dodatkowymi układami, których konstrukcja jest różna i zależna od numeru portu, jest układ sterowania tranzystorem wyjściowym ST, oraz układ polaryzacji tego tranzystora RX. Przerzutnik D jest elementem pola SFR i zawsze jest mu przypisany konkretny stan. Mówi się o "ustawieniu" przerzutnika, gdy na jego wyjściu Q obserwowany jest stan 1 logiczny oraz o stanie "skasowania" przerzutnika, gdy na jego wyjściu Q obserwowany jest stan 0 logicznego. Mikroprocesor może zmieniać stan przerzutnika poprzez zapis bezpośredni. Stan przerzutnika jest niezależny od stanu logicznego końcówki mikrokontrolera i może być odczytywany za pośrednictwem bramki trójstanowej nr 1. Stan końcówki mikrokontrolera może być odczytywany za pośrednictwem bramki trójstanowej nr 2. To, czy odczytywany jest stan przerzutnika czy też stan logiczny końcówki, zależy od wykonywanej instrukcji. Na przykład, do odczytywania stanu wszystkich końcówek portu P1 służy instrukcja *mov a,p1* (lub *mov c,p1.5* w przypadku odczytywania bitu). Dzięki możliwości odczytywania stanu przerzutnika, możliwe jest traktowanie tego elementu jako elementu pamięci. Umożliwia to wykonywanie na rejestrze portu (lub bitach tego rejestru) kilku operacji z grupy RMW, o których wspomniano w podrozdziale "Pole pamięci danych - RAM". Wykaz instrukcji RMW podano w tabeli 1.2.3 na str.16.



Rys. 1.2.11. Schemat struktury pojedynczej linii portu P0..P3.

Układ sterowania tranzystorem (ST) jest przełącznikiem, który umożliwia polaryzację bramki tranzystora, a tym samym powoduje jego włączenie lub wyłączenie. Sygnał sterowania bramką tranzystora może pochodzić od przerzutnika D albo od bitów innych elementów mikrokontrolera, np. z rejestrów PC, DPTR lub ACC. O pochodzeniu bitów sterowania decyduje stan linii *sterowanie* i *adres/dane*.

W portach P1, P2 i P3 układ polaryzacji jest identyczny i jest zbudowany z grupy tranzystorów, które swoim zachowaniem przypominają typowy rezystor obciążenia tranzystora - opornik polaryzacyjny (rys.1.2.12b,c,d). Ze względu na dołączenie tego opornika do "plusa" napięcia zasilania, VCC, nazywany jest on opornikiem podciągającym (pullup resistor). Jediną różnicą w stosunku do rzeczywistego rezystora jest to, że opór tego elementu jest zmienny i jest determinowany przez wewnętrzne sygnały sterowania mikrokontrolera. W czasie zmiany stanu końcówki, z wartości 0 na 1, opór rezystora gwałtownie maleje - przez okres dwóch taktów zegara opór zmniejsza się ponad 100-krotnie [1]. Takie rozwiązanie układu polaryzacji zapewnia szybkie przeładowanie pojemności pasożytniczych CX i szybkie osiągnięcie stanu wysokiego końcówki linii portu - stanu jedynki logicznej. W każdym innym przypadku, opór RX jest stosunkowo duży - jego przeciętna wartość wynosi ok. 40kΩ.

Port P0 jest pozbawiony opisanego wyżej opornika polaryzującego. Rolę układu polaryzującego spełnia w nim tranzystor TX (rys.1.2.12a), który przechodzi w stan przewodzenia wyłącznie w przypadku wykonywania funkcji alternatywnej trybu mikroprocesorowego (zapis/odczyt zewnętrznej pamięci) oraz deklarowania stanu wysokiego na końcówce linii. Przewodzenie tranzystora T definiuje stan niski końcówki linii portu - stan zera logicznego. Tranzystory TX i T, z powodów oczywistych, nigdy nie są załączone w tym samym czasie.

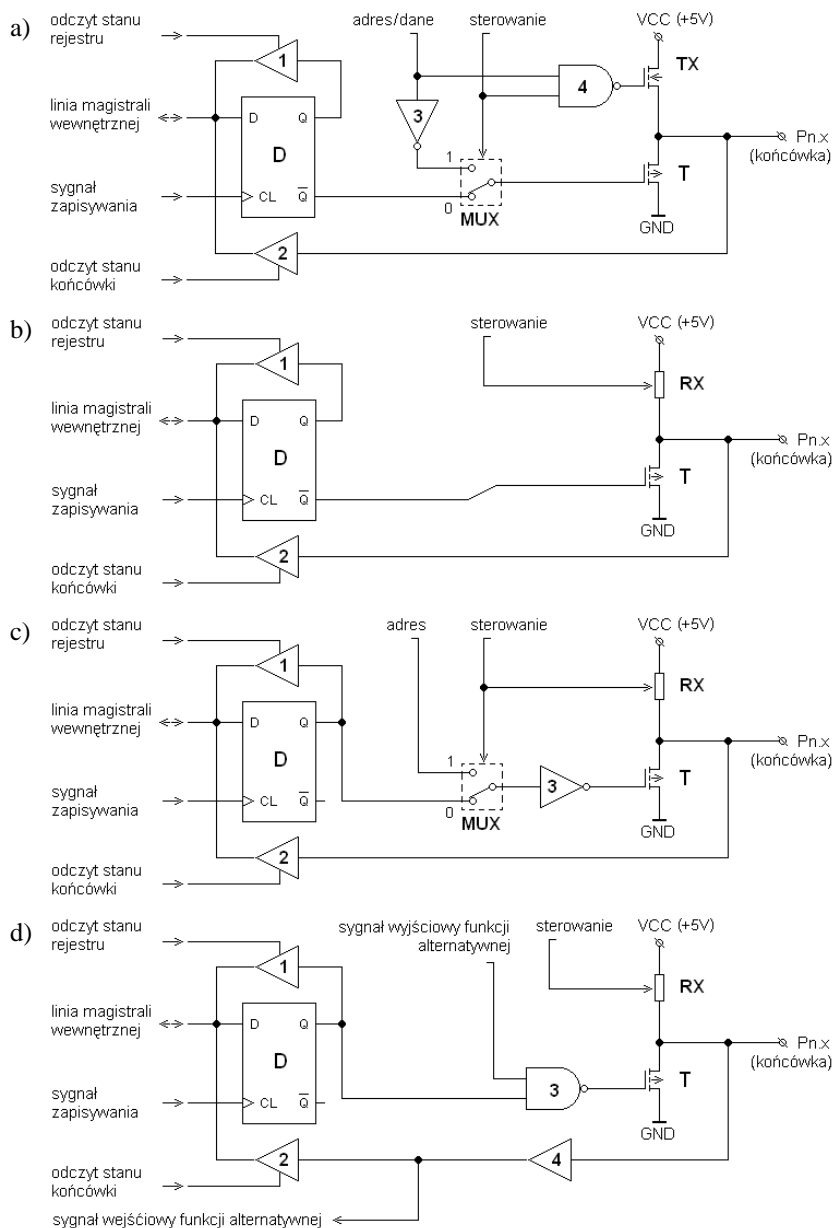
Tranzystor T, razem z układem polaryzacji RX, stanowią stopień wyjściowy linii portu i są one bezpośrednio dołączone do końcówki mikrokontrolera. Elementem czynnym tego stopnia jest tranzystor T. Bramka tranzystora może być sterowana różnymi sygnałami - najczęściej stosowany do tego celu jest sygnał stanu przerzutnika D. Ustawienie przerzutnika ($Q = 1$) powoduje wyłączenie tranzystora, a jego wykasowanie ($Q = 0$) - włączenie tranzystora. Dzięki obecności układu polaryzacji, w zależności od stanu przerzutnika, na końcówce pojawia się stan wysoki lub niski, 1 albo 0.

Jak już wspomniano, porty P0..P1 tworzą grupę tzw. portów uniwersalnych. Pojęcie uniwersalności wiąże się z możliwością zadeklarowania sposobu działania ich linii: mogą być liniami wejściowymi lub wyjściowymi. Deklaracja może dotyczyć wszystkich linii portu albo każdej linii z osobna. Ponadto, część linii portu (lub wszystkie linie tego portu) może być przełączona w tryb wykonywania funkcji alternatywnych, co wiąże się ze sprzętową deklaracją ich typu. Przy założeniu, że sygnałem sterowania tranzystora jest przerzutnik D, o sposobie używania linii będzie decydował stan tego przerzutnika:

- jeżeli linia portu ma być traktowana jako linia wyjściowa, to wpisanie do przerzutnika wartości 0 lub 1 spowoduje pojawienie się tej wartości na końcówce linii;
- w przypadku, gdy linia portu ma być traktowana jako linia wejściowa, o stanie końcówki będzie decydował urządzenie zewnętrzne; by nie dopuścić do tzw. konfliktu wyjść, tranzystor T nie może przewodzić - do przerzutnika D musi być wpisany stan 1.



status linii portu, wejście lub wyjście, jest uzależniony od stanu przerzutnika D - stan 0 zawsze deklaruje tryb wyjściowy z jednoczesnym ustawieniem stanu niskiego na tym wyjściu ..



Rys. 1.2.12. Struktura linii portów: P0 (a), P1 (b), P2 (c) oraz P3 (d).

Jak już wspomniano, nie tylko przerzutnik D może definiować stan wyjściowy linii. Mogą tego również dokonać sygnały funkcji alternatywnych. Np. praca w trybie mikroprocesorowym wymusza dostęp do pamięci zewnętrznej poprzez użycie portów P0, P2 oraz 2 linii z portu P3: P3.6 (WR) i P3.7 (RD). W czasie trwania wykonywania funkcji alternatywnej, w portach P0 i P2, o stanie wyjść poszczególnych linii decydują sygnały alternatywne. Podobnie dzieje się w porcie P3 z tym, że do poprawnego działania funkcji alternatywnej wymagane jest zadeklarowanie typu linii jako linii wejściowej.

Z dotychczasowego opisu wynika, że pomimo istnienia w każdej linii portu jednakowych elementów, o zachowaniu się linii poszczególnych portów decyduje układ sterowania tranzystorem T i sposób polaryzacji tego tranzystora. Podsumowując zauważone różnice, można stwierdzić, że:

port P0: tryb portu uniwersalnego:

- każda linia portu, w sposób indywidualny, może być ustawiona w stan wejścia lub wyjścia: ustawienie przerzutnika D definiuje tryb wejściowy; skasowanie przerzutnika D definiuje tryb wyjściowy z jednoczesnym ustaleniem stanu niskiego na końcówce linii;
- tranzystor TX nigdy nie przewodzi - po ustawieniu linii w stan wyjścia, w celu osiągnięcia stanu wysokiego na końcówce linii, niezbędne jest dołączenie do niej zewnętrznego opornika podciągającego;

tryb pracy alternatywnej:

- używany wyłącznie do wymiany informacji z pamięcią zewnętrzną lub zewnętrznymi urządzeniami I/O - wymiana informacji (przekazywanie młodszej części adresu oraz danej) odbywa się przy jednoczesnym wykorzystaniu wszystkich linii portu (patrz opis na str. 9);
- w trakcie wykonywania zapisywania informacji, tranzystory TX i T pracują w trybie komplementarnym - tylko jeden z nich jest w stanie przewodzenia;
- po ustawieniu przerzutników D, w momentach odczytywania informacji lub braku wymiany informacji, tranzystory TX i T są w stanie wyłączenia a końcówka linii nie jest spolaryzowana (jest w tzw. stanie trzecim) - dzięki tej właściwości, port P0 staje się typową dla systemów mikroprocesorowych magistralą dwukierunkową;

port P1: tryb portu uniwersalnego:

każda linia portu, w sposób indywidualny, może być ustawiona w stan wejścia lub wyjścia: ustawienie przerzutnika D definiuje tryb wejściowy lub wyjściowy ze stanem wysokim na końcówce linii;

skasowanie przerzutnika D definiuje tryb wyjściowy z jednoczesnym ustaleniem stanu niskiego na końcówce linii;

tryb pracy alternatywnej:

- dla 80C51 - brak;
- dla 80C52 - linie P1.0 i P1.1 są używane jako linie sterowania licznika T2:
 - P1.0 (T2) - wejście sygnału taktowania licznika T2;
 - P1.1 (T2EX) - wejście sygnału wyzwala licznika T2;

warunkiem koniecznym wykorzystania tych linii jest zadeklarowanie ich stanu jako stan wejściowy (przerzutnik D w stan 1);

port P2: tryb portu uniwersalnego:

identyczny z portem P1

tryb pracy alternatywnej:

używany wyłącznie przy wymianie informacji z pamięcią zewnętrzną lub zewnętrznymi urządzeniami I/O - wymiana informacji (przekazywanie starszej części adresu) odbywa się przy jednoczesnym wykorzystaniu wszystkich linii portu (patrz opis "Wymiana informacji z urządzeniami zewnętrznymi" na str. 9);

port P3: tryb portu uniwersalnego:

identyczny z portem P1

tryb pracy alternatywnej:

- każda linia portu jest związana z wewnętrznym urządzeniem I/O, stanowiąc dla niego kanał wejścia lub wyjścia - przyporządkowanie linii jest następujące:
 - P3.0 (RxD) - wejście portu transmisji szeregowej;
 - P3.1 (TxD) - wyjście portu transmisji szeregowej;
 - P3.2 (INT0) - wejście sygnału przerwania INT0 lub wejście bramkujące licznika T0;
 - P3.3 (INT1) - wejście sygnału przerwania INT1 lub wejście bramkujące licznika T1;
 - P3.4 (T0) - wejście sygnału taktowania licznika T0;
 - P3.5 (T1) - wejście sygnału taktowania licznika T1;
 - P3.6 (WR) - wyjście sygnału zapisu - strobu WR;
 - P3.7 (RD) - wyjście sygnału odczytu - strobu RD;
- funkcje alternatywne mogą być wykonane wyłącznie w przypadku wpisania do przerzutnika D stanu 1.

Jak już wspomniano, porty P1..P3 są elementami pola SFR - do każdej linii portu przypisany jest przerzutnik D. Z punktu widzenia mikroprocesora porty są rejestrami, umieszczonymi pod odpowiednimi adresami wewnętrznej pamięci

RAM. Wymiana informacji pomiędzy mikroprocesorem a portem odbywa się za pośrednictwem instrukcji adresowania bezpośredniego.

Na pokazanym niżej rysunku, przedstawiono strukturę rejestrów przypisanych portom P0..P3. W ramkach, symbolizujących 8-bitowe rejestry pamięci RAM, zaznaczono kratkami poszczególne bity rejestru - bit najstarszy jest umieszczony po lewej stronie ramki. Adres rejestru w polu SFR pokazany jest po prawej stronie ramki a adresy poszczególnych bitów pod właściwymi kratkami. Wewnątrz kratek umieszczono nazwy poszczególnych bitów.

P0:	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 2px;">P0.7</td> <td style="border: 1px solid black; padding: 2px;">P0.6</td> <td style="border: 1px solid black; padding: 2px;">P0.5</td> <td style="border: 1px solid black; padding: 2px;">P0.4</td> <td style="border: 1px solid black; padding: 2px;">P0.3</td> <td style="border: 1px solid black; padding: 2px;">P0.2</td> <td style="border: 1px solid black; padding: 2px;">P0.1</td> <td style="border: 1px solid black; padding: 2px;">P0.0</td> </tr> <tr> <td style="padding: 2px;">87h</td> <td style="padding: 2px;">86h</td> <td style="padding: 2px;">85h</td> <td style="padding: 2px;">84h</td> <td style="padding: 2px;">83h</td> <td style="padding: 2px;">82h</td> <td style="padding: 2px;">81h</td> <td style="padding: 2px;">80h</td> </tr> </table>	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	87h	86h	85h	84h	83h	82h	81h	80h	80h
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0											
87h	86h	85h	84h	83h	82h	81h	80h											
P1:	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 2px;">P1.7</td> <td style="border: 1px solid black; padding: 2px;">P1.6</td> <td style="border: 1px solid black; padding: 2px;">P1.5</td> <td style="border: 1px solid black; padding: 2px;">P1.4</td> <td style="border: 1px solid black; padding: 2px;">P1.3</td> <td style="border: 1px solid black; padding: 2px;">P1.2</td> <td style="border: 1px solid black; padding: 2px;">P1.1</td> <td style="border: 1px solid black; padding: 2px;">P1.0</td> </tr> <tr> <td style="padding: 2px;">97h</td> <td style="padding: 2px;">96h</td> <td style="padding: 2px;">95h</td> <td style="padding: 2px;">94h</td> <td style="padding: 2px;">93h</td> <td style="padding: 2px;">92h</td> <td style="padding: 2px;">91h</td> <td style="padding: 2px;">90h</td> </tr> </table>	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	97h	96h	95h	94h	93h	92h	91h	90h	90h
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0											
97h	96h	95h	94h	93h	92h	91h	90h											
P2:	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 2px;">P2.7</td> <td style="border: 1px solid black; padding: 2px;">P2.6</td> <td style="border: 1px solid black; padding: 2px;">P2.5</td> <td style="border: 1px solid black; padding: 2px;">P2.4</td> <td style="border: 1px solid black; padding: 2px;">P2.3</td> <td style="border: 1px solid black; padding: 2px;">P2.2</td> <td style="border: 1px solid black; padding: 2px;">P2.1</td> <td style="border: 1px solid black; padding: 2px;">P2.0</td> </tr> <tr> <td style="padding: 2px;">A7h</td> <td style="padding: 2px;">A6h</td> <td style="padding: 2px;">A5h</td> <td style="padding: 2px;">A4h</td> <td style="padding: 2px;">A3h</td> <td style="padding: 2px;">A2h</td> <td style="padding: 2px;">A1h</td> <td style="padding: 2px;">A0h</td> </tr> </table>	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	A7h	A6h	A5h	A4h	A3h	A2h	A1h	A0h	A0h
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0											
A7h	A6h	A5h	A4h	A3h	A2h	A1h	A0h											
P3:	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 2px;">P3.7</td> <td style="border: 1px solid black; padding: 2px;">P3.6</td> <td style="border: 1px solid black; padding: 2px;">P3.5</td> <td style="border: 1px solid black; padding: 2px;">P3.4</td> <td style="border: 1px solid black; padding: 2px;">P3.3</td> <td style="border: 1px solid black; padding: 2px;">P3.2</td> <td style="border: 1px solid black; padding: 2px;">P3.1</td> <td style="border: 1px solid black; padding: 2px;">P3.0</td> </tr> <tr> <td style="padding: 2px;">B7h</td> <td style="padding: 2px;">B6h</td> <td style="padding: 2px;">B5h</td> <td style="padding: 2px;">B4h</td> <td style="padding: 2px;">B3h</td> <td style="padding: 2px;">B2h</td> <td style="padding: 2px;">B1h</td> <td style="padding: 2px;">B0h</td> </tr> </table>	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	B7h	B6h	B5h	B4h	B3h	B2h	B1h	B0h	B0h
P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0											
B7h	B6h	B5h	B4h	B3h	B2h	B1h	B0h											

Rys. 1.2.13. Rejestry portów P0..P3 - wykaz adresów.

I jeszcze jedna uwaga. Po każdorazowym pojawieniu się aktywnego stanu sygnału kasowania mikrokontrolera, RST, wszystkie linie portów przechodzą w stan wejściowy - do wszystkich przerzutników D jest wpisywana jedynka logiczna.

1.2.5. Układ czasowo-licznikowy.

Układ czasowo-licznikowy (ang. timer/counter circuit) mikrokontrolera 80C51 składa się z dwóch liczników, T0 i T1, o maksymalnej pojemności 16 bitów każdy. Mikrokontroler 80C52, oprócz liczników T0 i T1, posiada jeszcze jeden licznik, T2 - również 16-bitowy.

Rozróżnienie pomiędzy funkcją czasomierza (ang. timer) a funkcją licznika (ang. counter) jest czysto umowne - w obu przypadkach jest nim licznik binarny, który zlicza wprowadzone do niego impulsy. Przyjęło się uważać, że czasomierz służy do wykreowania systemowej informacji o upływie zadanej jednostki czasu, a licznik jest przeznaczony do pozyskania informacji o liczbie wprowadzonych impulsów taktujących, przypadających na umowną jednostkę czasu (liczba zdarzeń w jednostce czasu). Z punktu widzenia sprzętowego, czasomierz jest dzielnikiem częstotliwości sygnału taktującego i wymaga taktowania sygnałem o stałej częstotliwości - do taktowania czasomierza najczęściej używa się sygnału zegarowego mikrokontrolera. Sygnał taktowania licznika zwykle

pochodzi z otoczenia sprzętowego mikrokontrolera i nie musi być sygnałem regularnym. Formalny podział układu licznikowego na czasomierz i licznik jest również podkreślany przez sposób obsługi tych urządzeń. W przypadku czasomierza, moment wykonania obsługi urządzenia narzuca samo urządzenie po fakcie przekroczenia zakresu zliczania. W przypadku realizowania funkcji licznika, moment wykonania jego obsługi (odczyt stanu licznika) jest zwykle determinowany przez inne urządzenie, np. inny układ czasomierza.

Budowa i działanie liczników T0 i T1.

Na rysunkach 1.2.14, 1.2.15 i 1.2.16 przedstawiono schematy blokowe liczników T0 i T1. Konstrukcja sprzętowa liczników jest identyczna, w związku z czym, na rysunkach, numer licznika ukryto pod indeksem i ($i = 0$ lub $i = 1$). Liczniki T0 i T1 są licznikami typu UP i zwiększają swój stan po napotkaniu elementu aktywnego sygnału taktującego - zliczają w górę. Elementem aktywnym sygnału taktowania jest jego zbocze opadające.

Każdy z liczników jest reprezentowany przez dwa rejestry 8-bitowe strefy SFR: TL i TH. Rejestr TL jest nazywany młodszym a rejestr TH starszym rejestrem licznika w przypadku, gdy tworzą razem rejestr 16-bitowy. Zarówno zapisywanie danej do tych rejestrów jak i odczytywanie odbywa się w sposób "bajtowy" - nie ma możliwości działań bitowych. Rejestrom TL i TH przydzielono następujące adresy:

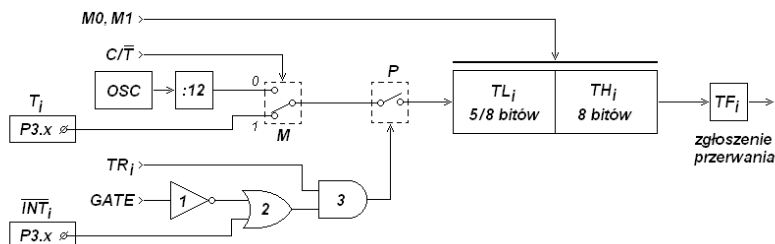
- TL0** - **8Ah** (mniej znaczący bajt licznika T0);
- TH0** - **8Ch** (bardziej znaczący bajt licznika T0);
- TL1** - **8Bh** (mniej znaczący bajt licznika T1);
- TH1** - **8Dh** (bardziej znaczący bajt licznika T1);

Rejestry TL i TH mogą pracować w różnych zestawieniach, nazywanych trybami pracy (ang. mode). O tym, jak dokonywane są wewnętrzne połączenia struktury, decyduje zestaw bitów, należących do tzw. rejestrów kontrolnych liczników, które również umieszczono w obrębie strefy SFR.

Dla każdego z liczników, T0 i T1, przydzielono po 4 bity kontrolne, które noszą nazwy: GATE, C/T, M1 i M0 i umieszczone są w rejestrze kontrolnym TMOD. Pozostałe bity kontrolno-sterujące, TF i TR, umieszczono w rejestrze kontrolnym TCON (dokładny opis bitów kontrolnych i ich adresy umieszczony jest za opisem budowy i działania liczników).

Tryb 0 i 1. Na rysunku 1.2.14 pokazano połączenia wewnętrznej struktury licznika dla trybu 0 i 1. Dla obu trybów połączenie głównych elementów licznika jest identyczne. Identyfikacyjny jest również sposób działania. Różnica pomiędzy trybami jest spowodowana wewnętrzną strukturą licznika **TL**: dla trybu 0 jest to licznik 5-bitowy natomiast dla trybu 1 - 8-bitowy. W trybie 0 wykorzystuje się 5 najmłodszych bitów licznika - stan bitów najstarszych, 5..7, nie jest brany pod uwagę. Biorąc pod uwagę obecność rejestru TH, można powiedzieć, że dla trybu

0 mamy do czynienia z licznikiem 13-bitowym, a w trybie 1 posługujemy się licznikiem 16-bitowym.



Rys. 1.2.14. Schemat blokowy liczników T0 i T1 pracujących w trybie 0 i 1.

Źródło sygnału taktowania licznika może być źródłem zewnętrznym lub wewnętrznym. O tym, które z nich ma być dostawcą sygnały taktowania, decyduje stan multiplexera (M), sterowanego bitem C/T . Gdy bit C/T jest wyzerowany, źródłem sygnału taktowania jest sygnał wewnętrzny. Do taktowania licznika używany jest sygnał o częstotliwości 12 razy mniejszej od częstotliwości sygnału zegarowego OSC , wytwarzanego przez generator mikrokontrolera.

Gdy bit C/T jest ustawiony na wartość 1, źródłem sygnału taktowania jest sygnał zewnętrzny, wprowadzony przez końcówkę T_i mikrokontrolera (T0 lub T1 - nazwa końcówki jest identyczna z nazwą licznika). O tym, czy licznik będzie zliczał czy też nie, decyduje stan przełącznika P , sterowanego bitami TR , $GATE$ i stanem końcówki mikrokontrolera INT . Sterowanie odbywa się za pośrednictwem bramek 1, 2 i 3. Wyzerowanie bitu TR , w każdym przypadku, blokuje możliwość zliczania. Ustawienie bitu TR umożliwia zliczanie w przypadku, gdy bit $GATE$ jest wyzerowany. Ustawienie bitu $GATE$ (gdy $TR = 1$) umożliwia zewnętrzne sterowanie zliczaniem licznika za pośrednictwem końcówki INT . Gdy stan wejścia $INT = 0$, licznik jest zablokowany; gdy $INT = 1$, licznik zlicza.

Tabela 1.2.5. Sterowanie zliczaniem impulsów zewnętrznych.

TR	GATE	INT	zliczanie
0	x	x	brak
1	0	x	jest
1	1	0	brak
1	1	1	jest

Jak już wspomniano, liczniki T0 i T1 są licznikami typu UP i po każdym zaobserwowaniu opadającego zbocza sygnału taktującego, zwiększają swój stan. Po przepełnieniu licznika, stan licznika ulega wyzerowaniu i jednocześnie ustawiany jest bit stanu przepełnienia, TF . Stan tego bitu jest flagą stanu licznika. Stan aktywny tego bitu, $TF=1$, powinien spowodować wywołanie programu obsługi zdarzenia związanego z licznikiem (patrz rozdz. 1.4.3). Wywołanie programu obsługi może nastąpić w wyniku testowania stanu bitu TF w obszarze pętli.

Ustawienie bitu TF może bezpośrednio wywołać program obsługi zdarzenia poprzez wygenerowanie przerwania. Przyjęcie przerwania automatycznie zeruje bit TF. Obsługa zdarzenia, realizowana bez używania przerw, nie zeruje automatycznie flagi TF - musi być ona skasowana programowo.



flaga przepełnienia licznika TF (TF0, TF1) jest kasowana w momencie przyjęcia przerwania ..

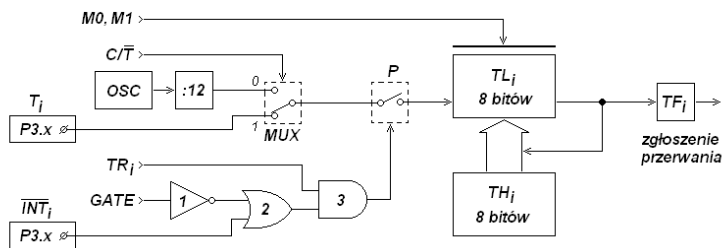
Programowa obsługa zdarzenia związanego z licznikiem jest wykonywana po jego przepełnieniu, gdy stan licznika jest zerowany. Obsługa sprowadza się najczęściej do wpisania do licznika danej określającej stan początkowy zliczania. Im dana jest większa tym mniej impulsów taktowania jest potrzebnych do ponownego przepełnienia licznika. Różnica pomiędzy pojemnością licznika a wielkością danej określa wskaźnik modulo licznika (wskaźnik podziału częstości).



częstość zdarzeń generowanych przez czasomierz jest definiowana przez stan początkowy licznika ..

Stan rejestrów licznika może być odczytywany i zmieniany w każdym momencie, bez względu na to, czy licznik zlicza czy też nie. Ponieważ dostęp do licznika jest dwuetapowy (2 rejestry, TL i TH), w przypadku gdy licznik pracuje, jakakolwiek próba, np. odczytania stanu licznika, może się zakończyć wynikiem obciążonym dużym błędem. Jest to związane z tym, że kolejne odczyty mogą być związane z różnym stanem licznika. Taką sytuację należy przewidzieć - więcej szczegółów na ten temat znajduje się w części poświęconej programowej obsłudze liczników.

Tryb 2. Na rysunku 1.2.15 pokazano połączenia wewnętrznej struktury licznika pracującego w trybie 2. Tym razem licznikiem jest 8-bitowy rejestr TL, który pracuje w trybie autoregeneracji (ang. auto-reload mode). W momencie przekroczenia pojemności licznika TL, następuje jego automatyczne uzupełnienie daną, przechowywaną w rejestrze TH. Przekroczenie stanu licznika powoduje również ustawienie bitu flagowego TF, mogącego wywołać przerwanie. Częstość pojawiania się stanu aktywnego bit TF jest uzależniona od wartości danej wpisanej do rejestru TH. Im dana większa, tym częstsze ustawianie bitu TF. W przypadku granicznym, gdy dana jest równa 255, ustawienie flagi TF występuje co 12 cykli zegarowych mikrokontrolera (co 1 cykl maszynowy). Stan rejestru TH może być odczytywany i modyfikowany w dowolnym momencie. Sposób sterowania pracą licznika TL jest identyczny jak dla trybu 0 i 1.



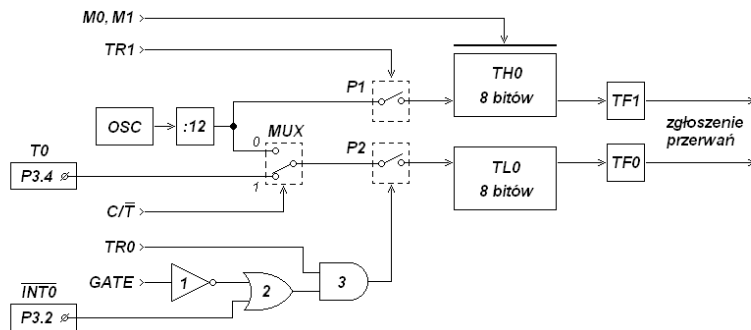
Rys. 1.2.15. Schemat blokowy liczników T0 i T1 pracujących w trybie 2.

Tryb 2 pracy licznika jest bardzo wygodny do odmierzenia krótkich odcinków czasu w systemie mikroprocesorowym, np. sygnału taktowania elementów portu transmisji szeregowego (licznik T1) lub do wytwarzania tzw. czasu podstawowego systemu (zadanie laboratoryjne 1).



licznik T1 może być generatorem sygnału zegarowego dla portu transmisji szeregowej - najlepiej do tego celu nadaje się tryb 2 pracy licznika ..

Tryb 3. Na rysunku 1.2.16 pokazano połączenia wewnętrznej struktury licznika pracującego w trybie 3. Wybór tego trybu dla licznika T1 skutkuje jego zatrzymaniem. Rejestry TL0 i TH0 licznika T0 stają się dwoma niezależnymi, 8-bitowymi licznikami.



Rys. 1.2.16. Schemat blokowy licznika T0 pracującego w trybie 3.

Licznik TH0 może być taktowany sygnałem o częstotliwości 12 razy mniejszej od częstotliwości zegarowej mikrokontrolera. O tym, czy licznik pracuje czy też nie, decyduje stan bitu TR1. Po przepelnieniu licznika, stan licznika ulega wyzerowaniu i ustawiany jest bit flagowy TF1, mogący wygenerować przerwanie.

Licznik TL0 może być taktowany tym samym sygnałem co licznik TH0 lub sygnałem zewnętrznym, pobranym z końcówki T0 mikrokontrolera. O wyborze źródła taktowania decyduje stan bitu C/T - gdy bit C/T jest wyzerowany, źródłem jest sygnał wewnętrzny. O tym, czy licznik pracuje czy też nie, decyduje stan bitów TR0, GATE oraz INT0. Sposób sterowania pracą licznika TL jest

identyczny jak dla trybu 0 i 1 (patrz tabela 1.2.5). Po przepelnieniu licznika, stan licznika ulega wyzerowaniu i ustawiany jest bit flagowy TF0, mogący wygenerować przerwanie.

Budowa i działanie licznika T2 w 80C52.

Licznik T2 jest stałym elementem rdzenia mikrokontrolera 80C52. Podobnie jak liczniki T0 i T1, może on pełnić funkcje czasomierza lub licznika impulsów zewnętrznych. Jego konstrukcja jest zdecydowanie bardziej zaawansowaną w stosunku do liczników T0 i T1 ale, podobnie jak w przypadku liczników T0 i T1, jest oparta o 16-bitowy licznik typu UP. Stan licznika jest zwiększany przez opadające zbocze sygnału taktowania. Licznik jest złożony z dwu, 8-bitowych rejestrów, TL2 i TH2. Rejestr TL nazywany jest młodszym rejestrem licznika T2 a rejestr TH starszym. W odróżnieniu od liczników T0 i T1, licznik T2 zawsze pracuje w konfiguracji 16-bitowej.

Z licznikiem T2 ściśle współpracuje 16-bitowy rejestr RCAP, który jest zbudowany z 2 rejestrów 8-bitowych: rejestru młodszego, RCAPL i starszego, RCAPH. Rejestr RCAP spełnia w stosunku do licznika T2 funkcję rejestru pamięciowego. W zależności od trybu pracy licznika T2, do rejestru RCAP można przepisać pełną zawartość licznika albo odwrotnie - daną z rejestru RCAP można przepisać do licznika. Każdy cykl zapisu czy odczytu jest realizowany sprzętowo i dotyczy słowa 16-bitowego.

Rejestrom TL2, TH2, RCAPL i RCAPH, które umieszczone są w strefie SFR, przydzielono następujące adresy:

- TL2** - **CCh** (mniej znaczący bajt licznika T2);
- TH2** - **CDh** (bardziej znaczący bajt licznika T2);
- RCAPL** - **CAh** (mniej znaczący bajt rejestru pomocniczego licznika T2);
- RCAPH** - **CBh** (bardziej znaczący bajt rejestru pomocniczego liczn. T2);

Zapisywanie danej do tych rejestrów jak i odczytywanie odbywa się w sposób "bajtowy" - nie ma możliwości działań bitowych.

O tym, w jakiej konfiguracji pracują elementy licznika T2 i jak są sterowane, decydują bity sterowania zgromadzone w rejestrze T2CON, umieszczonym w strefie SFR. Bitami kontrolnymi licznika T2 są: CP/RL2, C/T2, TR2, EXEN2, TCLK, RCLK, EXF2 i TF2. (dokładny opis bitów kontrolnych i ich adresy umieszczony jest za opisem budowy i działania licznika T2).

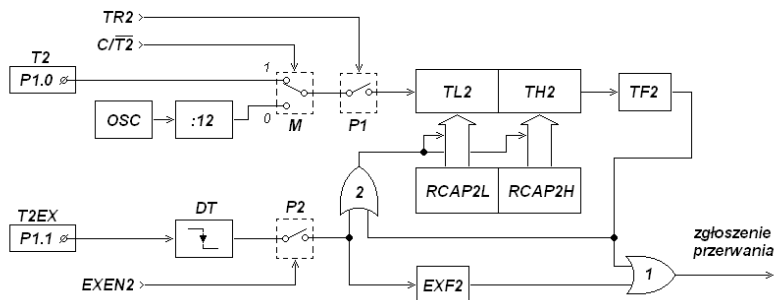
Tryb pracy licznika T2 jest ustalany przez bity CP/RL2, C/T2 oraz sumę logiczną bitów RCLK i TCLK. Stan zera logicznego relacji jest osiąganym w przypadku gdy RCLK=0 i TCLK=0; w każdym innym przypadku stanem relacji jest jedynka logiczna. O tym, czy licznik pracuje czy też nie, decyduje stan bitu TR2. Poszczególne trybom pracy licznika T2 przypisano kolejne nazwy: tryb przechwytywania (ang. Capture Mode), tryb autoprzeladowania (ang. Auto-Reload Mode) oraz tryb generatora szybkości transmisji (ang. Baud Rate Generator

Mode). Zestawienie trybów pracy w funkcji stanu bitów sterujących podano w tabeli 1.2.6.

Tabela 1.2.6. Sterowanie trybem pracy licznika T2.

RCLK TCLK	CP/RL2	C/T2	tryb pracy / źródło taktowania
0	0	0	Auto-Reload / generator wewnętrzny
0	0	1	Auto-Reload / sygnał zewnętrzny
0	1	0	Capture / generator wewnętrzny
0	1	1	Capture / sygnał zewnętrzny
1	x	0	Baud Rate Generator / sygnał wewnętrzny
1	x	1	Baud Rate Generator / sygnał zewnętrzny

Tryb autoprzeladowania. Na rysunku 1.2.17 pokazano połączenia wewnętrznej struktury licznika T2 pracującego w trybie autoprzeladowania. Ten tryb pracy jest wybierany przez wyzerowanie bitów CP/RL2, RCLK i TCLK. Praca w trybie autoprzeladowania polega na sprzętowym przepisaniu stanu 16-bitowego rejestru pomocniczego RCAP do licznika. Jest to działanie analogiczne do obsługi liczników T0 i T1 w trybie 2.



Rys. 1.2.17. Schemat blokowy licznika T2 pracującego w trybie autoprzeladowania.

Źródło sygnału taktowania licznika może być źródłem zewnętrznym lub wewnętrznym. O tym, które z nich ma być dostawcą sygnały taktowania, decyduje stan multiplexera (M), sterowanego bitem C/T2. Gdy bit C/T2 jest ustawiony na wartość 1, źródłem jest sygnał zewnętrzny. Gdy bit C/T2 jest wyzerowany, źródłem jest sygnał wewnętrzny. W przypadku źródła zewnętrznego, sygnał taktowania musi być wprowadzony przez końcówkę mikrokontrolera, T2 (P1.0). Źródłem wewnętrznym jest generator sygnału zegarowego mikrokontrolera, OSC. Podobnie jak w licznikach T0 i T1, do taktowania licznika używany jest sygnał o częstotliwości 12 razy mniejszej od częstotliwości sygnału zegarowego - jest on wytwarzany w układzie dzielnika częstotliwości.

O tym, czy licznik będzie zliczał czy też nie, decyduje stan przełącznika P1, sterowanego bitem TR2. Wyzerowanie bitu TR2 blokuje możliwość zliczania, a jego ustawienie, umożliwia zliczanie. Przepelnienie licznika jest sygnalizowane przez ustawienie bitu flagowego TF2, co z kolei może być przyczyną wygenerowania przerwania. W momencie przepelnienia licznika, w sposób automatyczny, do licznika jest wpisywana zawartość rejestru RCAP - następuje proces autoprzeladowania licznika. W odróżnieniu od liczników T0 i T1, flaga stanu licznika, TF2, nie jest automatycznie kasowana po fakcie przyjęcia przerwania.



flaga przepelnienia licznika, TF2, nie jest kasowana w momencie przyjęcia przerwania - wymagane jest jej skasowanie w ramach programu obsługi zdarzenia ..

Dodatkową opcję przepisywania danej do licznika oferuje stan bitu kontrolnego, EXEN2. Gdy EXEN2=0 to w momencie przepelnienia licznika następuje wyżej opisany proces autoprzeladowania. W przypadku, gdy EXEN2=1, przepisywanie automatyczne jest dalej kontynuowane ale dodana jest nowa opcja. Przepisanie danej do licznika może być również dokonane przez opadające zboczne sygnału zewnętrznego, wprowadzonego do mikrokontrolera przez końcówkę T2 (P1.0). Fakt przepisywania sygnałem zewnętrznym jest rejestrowany poprzez ustawienie bitu flagowego, EXF2, co może być przyczyną wygenerowania przerwania.



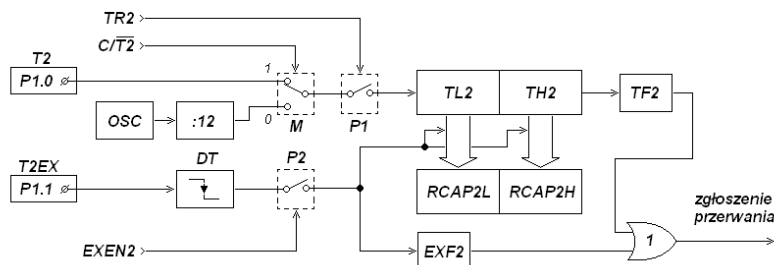
flaga przepelnienia licznika, EXF2, nie jest kasowana w momencie przyjęcia przerwania - wymagane jest jej skasowanie w ramach programu obsługi zdarzenia ..

Informacja o wystąpieniu przerwania w strefie licznika T2 jest przekazywana do układu kontrolera przerw za pośrednictwem pojedynczej linii sygnałowej. Sygnał przerwania jest sumą logiczną stanu bitów flagowych, TF2 i EXF2 (patrz bramka nr 1 na rys.1.2.17). Gdy obydwa źródła przerwania są aktywne to w początkowej fazie wykonywania programu obsługi przerwania powinno się zbadać stan bitów flagowych i dopiero na tej podstawie realizować dalszą część programu. W momencie kończenia programu obsługi, obydwa bity flagowe, T2F i EXF2, muszą być skasowane.

Tryb przechwytywania.

Na rysunku 1.2.18 pokazano wewnętrzną strukturę licznika T2 pracującego w trybie przechwytywania. Ten tryb pracy jest wybie-rany przez ustawienie bitu CP/RL2 i wyzerowanie bitów RCLK i TCLK. Praca w trybie przechwytywania pozwala na sprzętowe przepisanie stanu 16-bitowego licznika do rejestru pomocniczego RCAP. Pozwala to na np. precyzyjny pomiar parametrów sygnału zewnętrznego.

O wyborze źródła sygnału taktującego i sposobie zarządzania pracą licznika decydują bity TR2 i C/T2 w sposób identyczny jak w trybie przeładowania. Tak jak w poprzednio opisanym trybie, fakt przepełnienia licznika jest sygnalizowany ustawieniem bitu flagowego TF2, co może być przyczyną wygenerowania przerwania.



Rys. 1.2.18. Schemat blokowy licznika T2 pracującego w trybie przechwytywania.

Jak już wspomniano, ideą trybu przechwytywania jest możliwość sprzętowego przepisania stanu licznika do jego rejestru pomocniczego, RCAP. Inicjatorem momentu przepisywania jest zmiana stanu sygnału zewnętrznego, wprowadzonego do mikrokontrolera przez jego końcówkę T2EX (P1.1). Sygnał przepisywania jest badany w układzie detektora DT - opadające zbocze sygnału zewnętrznego może spowodować przepisanie stanu licznika. O tym, czy przepisanie nastąpi czy też nie, decyduje stan przełącznika P2, sterowanego bitem kontrolnym EXEN2. Gdy EXEN2=0, możliwość przepisywania jest zablokowana. Gdy EXEN2=1, opadające zbocze sygnału zewnętrznego powoduje przepisanie bieżącego stanu licznika T2 do rejestru RCAP. Fakt przepisywania jest rejestrowany poprzez ustawienie bitu flagowego EXF2, co może być przyczyną wygenerowania przerwania.

Tryb generatora szybkości transmisji.

Na rysunku 1.2.19 pokazano połączenia wewnętrznej struktury licznika T2 pracującego w trybie generatora. Ten tryb jest przeznaczony do wytworzenia sygnału taktującego (zegarowego) dla odbiornika i nadajnika układu transmisji szeregowej. Tryb pracy generatora jest wybierany przez ustawienie któregośkolwiek z bitów RCLK lub TCLK - stan bitu CP/RL2 jest wtedy nieistotny.

Porównując schematy blokowe licznika T2, przypisane trybowi generatora i trybowi autoprzeładowania widać, że konfiguracja elementów licznika jest bardzo podobna. Wybór źródła sygnału taktującego dla licznika i sposób zarządzania jego pracą jest taki sam jak dla trybów opisanych wyżej. Po przepełnieniu licznika następuje proces autoprzeładowania - do licznika jest wpisywana dana z rejestru RCAP. W odróżnieniu od trybu autoprzeładowania, w trybie generatora nie jest ustawiana flaga TF2 - przeładowanie licznika nie może być przyczyną generacji przerwania.

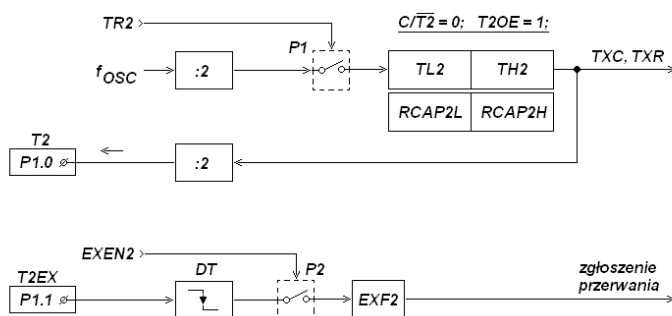
generowanie przerwania. Pozwala to na potraktowanie końcówki P1.0 jako miejsce wprowadzania do mikrokontrolera następnego sygnału przerwania zewnętrznego (następnego po INT0 i INT1). Do włączania lub wyłączania aktywności tego toru przeznaczony jest przełącznik P2 sterowany bitem EXEN2. Ustawienie bitu aktywuje tor a wyzerowanie bitu blokuje możliwość wygenerowania przerwania.

Budowa i działanie licznika T2 w AT89S8253.

Licznik T2 mikrokontrolera AT89S8253, w stosunku do układu wzorcowego, 80C52, jest uzupełniony o dwa dodatkowe tryby pracy. Tryby te można uaktywnić za pośrednictwem dwu bitów kontrolnych, T2OE i DCEN, które umieszczono w rejestrze T2MOD ze strefy SFR. Dostęp do bitów jest możliwy instrukcjami adresowania bezpośredniego, słowem 8-bitowym. (dokładny opis bitów kontrolnych i ich adresy umieszczono za opisem budowy i działania licznika T2).

Tryb generatora zegarowego.

Uproszczony schemat blokowy elementów licznika, pracującego w trybie generatora zegarowego (ang. Clock Out Mode), pokazano na rysunku 1.2.20. Tryb ten jest definiowany przez ustawienie bitu T2OE ($T2OE=1$). Licznik jest taktowany sygnałem zegarowym mikrokontrolera o zmniejszonej dwukrotnie częstotliwości. Zliczający w górę licznik T2, po przepełnieniu jest automatycznie przeładowywany zawartością rejestru pomocniczego RCAP. Wytworzony w liczniku T2 sygnał periodyczny, po dwukrotnym obniżeniu jego częstotliwości jest wyprowadzony przez końcówkę mikrokontrolera, T2 (P1.0). Wyprowadzany sygnał ma wypełnienie 50% a jego częstotliwość jest definiowana przez daną umieszczoną w rejestrze RCAP. Ze względu na kształt sygnału, opisywany tryb pracy licznika T2 często nazywany jest generatorem fali prostokątnej.



Rys. 1.2.20. Uproszczony schemat blokowy licznika T2 pracującego w trybie generatora fali prostokątnej.

Flaga TF2 w opisywanym trybie jest nieużywana i przepełnienie licznika nie może być zgłoszone przerwaniem. Identycznie jak w trybie generatora szybkości transmisji, przerwanie może być wygenerowane wyłącznie przez opadające zboczne sygnału zewnętrznego, wprowadzonego przez końcówkę T2EX.

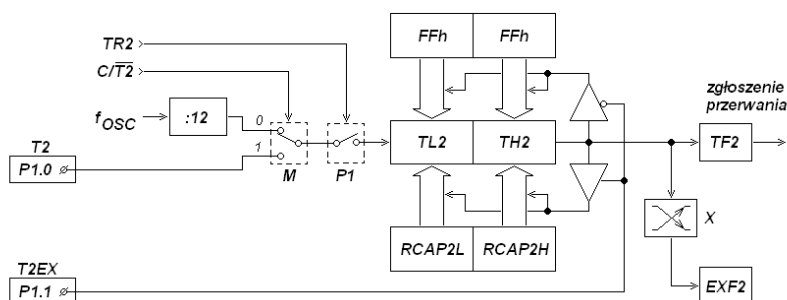
Zakładając, że pod oznaczeniem *RCAP* znajduje się dana umieszczona w rejestrze *RCAP*, częstotliwość generowanej fali prostokątnej określa się wzorem:

$$f_{out} = \frac{f_{osc}}{4 * (65536 - RCAP)}$$

Oprócz generowania fali prostokątnej, licznik może być używany do taktowania elementów portu transmisji szeregowej. Częstotliwość sygnału fali prostokątnej, w takim przypadku, jest zdeterminowana przez szybkość transmisji.

Tryb licznika rewersyjnego.

Drugi ze wspomnianych trybów umożliwia pracę licznika zarówno w trybie UP jak i DOWN - umożliwia zliczanie w górę i w dół. Ponadto, dla obu opcji zapewniony jest proces automatycznego przeładowania stanu licznika po osiągnięciu wartości granicznych. Ze względu na sposób działania, przypisano omawianemu trybowi uproszczoną nazwę: trybu licznika rewersyjnego (ang. Auto Reload Up or Down Counter Mode). Schemat blokowy konfiguracji elementów licznika pokazano na rysunku 1.2.21.



Rys. 1.2.21. Schemat blokowy licznika T2 pracującego w trybie licznika rewersyjnego z automatycznym przeładowaniem stanu licznika.

Wybór trybu licznika rewersyjnego jest dokonywany przez ustawienie bitu kontrolnego DCEN (DCEN=1), umieszczonego w rejestrze T2MOD. Kierunek zliczania, w górę lub dół, jest wybierany przez stan logiczny końcówki T2EX (P1.1) mikrokontrolera. Stan niski końcówki, T2EX=0, definiuje zliczanie w dół a stan wysoki, T2EX=1, zliczanie w górę. Źródłem sygnału taktującego licznik T2 może być sygnał zegarowy mikrokontrolera, którego częstotliwość została zmniejszona 12 razy albo sygnał zewnętrzny, wprowadzony na końcówkę T2 (P1.0). O wyborze źródła taktowania decyduje stan bitu C/T2. Gdy C/T2=0 to źródłem sygnału taktowania jest generator wewnętrzny. Gdy C/T2=1, źródłem jest sygnał zewnętrzny.

Jak już wspomniano, warunkiem autoprzeładowania stanu licznika jest osiągnięcie wartości granicznych. W przypadku zliczania w górę, wartością graniczną jest stan przekroczenia pojemności licznika. W takim przypadku, do licznika jest przepisywana zawartość rejestru *RCAP*.

W przypadku zliczania w dół, wartością graniczną jest stan licznika identyczny z zawartością rejestru RCAP. W takim przypadku, do licznika jest wprowadzana wartość 65535 (0FFFFh).

Każdy cykl automatycznego przeładowania stanu licznika, bez względu na to czy licznik zlicza w górę czy też w dół, powoduje ustawienie bitu TF2 oraz zmianę stanu bitu flagowego EXF2 na stan przeciwny. Bit TF2 może spowodować wywołanie przerwania. Bit EXF2 nie generuje przerwania - można go potraktować jako 17 z kolei element 16-bitowego licznika.

Rejestry i bity kontrolne układu czasowo licznikowego.

Jak już wspomniano, o sposobie pracy liczników układu czasowo-licznikowego decyduje stan grupy bitów, rozmieszczonych w kilku rejestrach kontrolnych, w strefie SFR. Dostęp do tych rejestrów jest możliwy za pomocą instrukcji adresowania bezpośredniego. Sposób pracy i stan początkowy liczników powinien być ustalony przed pierwszym ich wykorzystaniem. Czynności te powinny być wykonane w pierwszej fazie programu - w fazie przygotowania środowiska (patrz rozdział 1.4.2).

Bity kontrolne liczników T0 i T1.

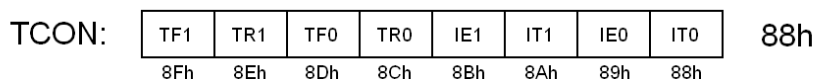
Do sterowania pracą liczników T0 i T1 służą bity kontrolne, zgrupowane w rejestrach TMOD i TCON. Rejestr TCON może być adresowany bitowo.



Rys. 1.2.22. Rejestr kontrolny TMOD.

Rejestr TMOD jest przeznaczony do określenia trybu i sposobu pracy liczników T0 i T1. Licznikowi T0 przypisano 4 młodsze bity a licznikowi T1 - cztery starsze bity rejestru. Nazwy odpowiadających sobie bitów z każdej podgrupy są identyczne. Opis działania liczników T0 i T1 w funkcji stanu bitów ujęto w tabeli 1.2.7.

W rejestrze TCON zgromadzono bity flagowe stanu przepełnienia licznika, TF0 i TF1 oraz bity sterowania pracą liczników, TR0 i TR1. Pozostałe bity, IE1, IT1, IE0 oraz IT0 są zarezerwowane do obsługi przerwania i zostaną omówione w rozdziale 1.2.7 (podrozdział "Rejestry i bity sterujące układu kontrolera przerwania", str. 55).



Rys. 1.2.23. Rejestr kontrolny TCON.

Gdy bit TR odpowiedniego licznika jest wyzerowany (TR=0) to licznik jest w stanie pasywnym - nie zlicza. Gdy bit TR jest ustawiony (TR=1), licznik pracuje.

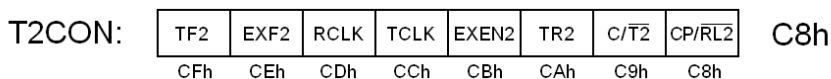
Gdy stan zliczeń przekroczy pojemność licznika, odpowiadający licznikowi bit flagowy TF zostanie ustawiony. Jeżeli uaktywnione są przerwania, w momencie przyjęcia przerwania bit TF jest automatycznie zerowany. W przypadku pracy z wyłączonymi przerwaniami, bit TF musi być wyzerowany programowo.

Tabela 1.2.7. Sterowanie pracą liczników T0 i T1.

bit(y)	stan	opis działania
GATE	0	sygnał zewnętrznego bramkowania (INT) - zablokowany
	1	sygnał zewnętrznego bramkowania (INT) - aktywny
C/T	0	wewnętrzny sygnał taktowania (funkcja czasomierza)
	1	zewnętrzny sygnał taktowania (funkcja licznika)
M1, M0	00	praca w trybie 0
	01	praca w trybie 1
	10	praca w trybie 2
	11	praca w trybie 3

Bity kontrolne licznika T2.

Do sterowania pracą licznika T2 w mikrokontrolerze 80C52 służą bity kontrolne, zgrupowane w rejestrze T2CON. Rejestr T2CON może być adresowany bitowo.



Rys. 1.2.24. Rejestr kontrolny T2CON.

Działanie poszczególnych bitów rejestru jest następujące:

- CP/RL2** - ustawianie trybu pracy licznika gdy RCLK=0 i TCLK=0:
 0 - praca w trybie automatycznego przeładowania;
 1 - praca w trybie przechwytywania;
- C/T2** - wybór źródła sygnału taktowania licznika:
 0 - sygnał generatora wewnętrzny;
 1 - sygnał zewnętrzny (końcówka T2);
- TR2** - sterowanie procesem zliczania licznika:
 0 - zliczanie zablokowane;
 1 - zliczanie odblokowane;
- EXEN2** - aktywowanie wejścia T2EX (P1.1):
 0 - wejście T2EX zablokowane;
 1 - wejście T2EX w stanie aktywnym;

- TCLK** - dołączanie licznika T2 do nadajnika portu transmisji szeregowej:
 0 - brak sprzężenia;
 1 - nadajnik jest taktowany przez licznik T2;
- RCLK** - dołączanie licznika T2 do odbiornika portu transmisji szeregowej:
 0 - brak sprzężenia;
 1 - odbiornik jest taktowany przez licznik T2;
- EXF2** - bit flagowy stanu sygnału zewnętrznego, doprowadzonego do końcówki T2EX mikrokontrolera (P1.1):
 0 - stan pasywny - nic się nie dzieje;
 1 - nastąpiło wykrycie opadającego zbocza sygnału;
- TF2** - bit flagowy stanu licznika:
 0 - stan pasywny - nic się nie dzieje;
 1 - nastąpiło przepełnienie licznika - w ramach programu obsługi zdarzenia, bit TF2 musi być wyzerowany programowo.

W mikrokontrolerze AT89S8253, do sterowania pracą licznika T2 służą bity kontrolne opisanego wyżej rejestru T2CON oraz 2 dodatkowe bity, umieszczone w rejestrze T2MOD. Rejestr T2MOD nie może być adresowany bitowo.

T2MOD:

-	-	-	-	-	-	T2OE	DCEN
---	---	---	---	---	---	------	------

 C9h

Rys. 1.2.25. Rejestr kontrolny T2MOD.

Działanie poszczególnych bitów rejestru jest następujące:

- DCEN** - aktywowanie trybu licznika rewersyjnego:
 0 - licznik wyłącznie w trybie UP;
 1 - praca w trybie licznika rewersyjnego;
- T2OE** - wybór trybu generatora fali prostokątnej:
 0 - praca w trybie standardowym (80C52);
 1 - praca w trybie generatora fali prostokątnej - sygnał generatora jest wyprowadzony przez końcówkę T2 (P1.0).

W tabeli 1.2.8 pokazano zestawienie czynności wykonywanych przez licznik T2 w funkcji stanu bitów kontrolnych i typu mikrokontrolera. Licznik mikrokontrolera AT89S8253 jest w stanie zrealizować wszystkie opcje tabeli; licznik układu 80C52 - tylko wskazane. Stan bitu, opisany znakiem *x*, jest nieistotny dla danego trybu pracy. Brak definicji typu końcówki T2 oznacza, że może ona być standardową linią portu P1.

Dla wszystkich ujętych w tabeli przypadków, ustawienie bitu TR2 (TR2=1) pociąga za sobą rozpoczęcie pracy licznika. Wyzerowanie bitu TR2 powoduje zatrzymanie pracy licznika.

Tabela 1.2.8. Sterowanie pracą licznika T2.

bity kontrolne					tryb pracy - sposób wykonywania pracy										
RCLK + TCLK	CP/RL2	C/T2	T2OE	DCEN	mikrokontroler 82C52	licznik	generator	przetładowanie	przechwytywanie	taktowanie sygnałem wewnętrznym	podzielnik preskalera	taktowanie sygnałem zewnętrznym	przerwanie - TF2	przerwanie - EXF2	typ końcówki T2
0	0	0	0	0	•	•		•		•	12		•	•	I/O
0	0	1	x	0	•	•		•			12	•	•	•	we
0	1	0	0	0	•	•			•	•	12		•	•	I/O
0	1	1	x	0	•	•			•		12	•	•	•	we
0	0	0	x	1		•		•		•	12		•		we
0	0	1	x	1		•		•			12	•	•		we
1	x	0	0	x	•		•	•		•	2			•	I/O
1	x	1	x	x	•		•	•			2	•		•	we
1	x	0	1	x			•	•		•	2			•	wy

1.2.6. Port transmisji szeregowej.

Mikrokontrolery z rodziny MCS-51 są wyposażone w uniwersalny port transmisji szeregowej, umożliwiający synchroniczną lub asynchroniczną wymianę informacji z innym komputerem. Do wymiany informacji port szeregowy używa 2 linie, RXD oraz TXD, które są liniami portu P3, odpowiednio P3.0 oraz P3.1. W przypadku wykorzystywania portu szeregowego, linie te nie mogą być używane jako linie uniwersalne i powinny być ustawione w stan wejściowy (lub wyjściowy ze stanem 1). Jeżeli port transmisji szeregowej jest aktywowany ale w danym momencie nie zachodzi wymiana informacji to linie RXD i TXD są zawsze ustawione w stan jedynki logicznej.

Port transmisji szeregowej może pracować w 4 trybach, przy czym tryb 0 jest dedykowany transmisji synchronicznej, a pozostałe, 1..3, transmisji asynchronicznej. Wysyłanie i odbieranie danych, które jest realizowane transmisją asynchroniczną, odbywa się w sposób całkowicie niezależny od siebie (ang. full-duplex). W przypadku transmisji synchronicznej, ze względu na konieczność istnienia linii zegarowej, wymiana informacji odbywa się za pomocą 1 linii - wysyłanie i odbieranie danych odbywa się w sposób naprzemienny (ang. half-duplex).

Tabela 1.2.9. Tryby pracy portu transmisji szeregowej.

tryb	typ transmisji	liczba bitów danej	szybkość transmisji	częstotliwość sygnału taktowania
0	synchroniczny	8	stała	$f_{osc}/12$
1	asynchroniczny	8	regulowana	nastawa licznika T1 lub T2
2	asynchroniczny	9	2 szybkości	$f_{osc}/32$ lub $f_{osc}/64$
3	asynchroniczny	9	regulowana	nastawa licznika T1 lub T2

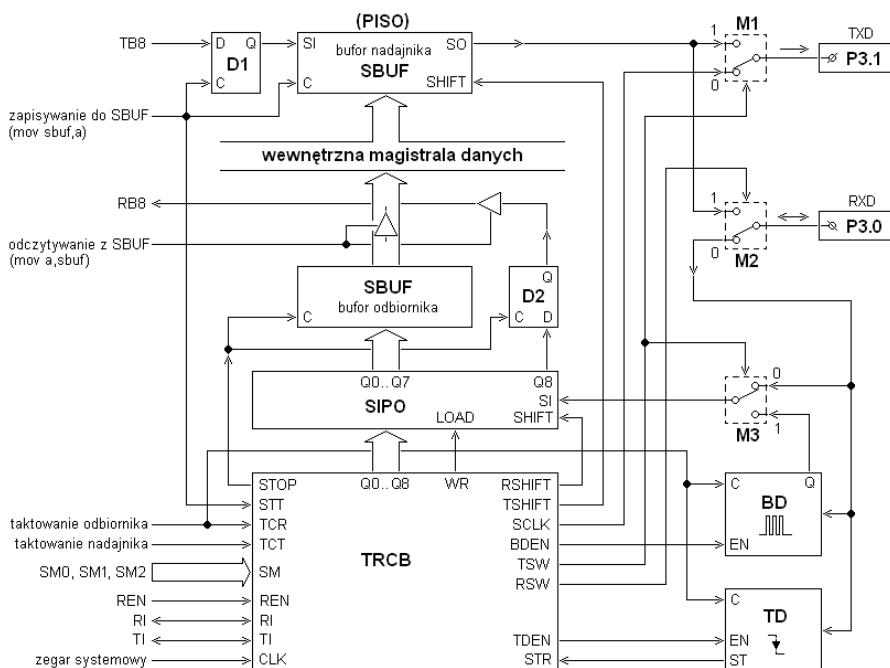
Port transmisji szeregowej jest reprezentowany przez 2 rejestry strefy SFR, które noszą nazwy SCON oraz SBUF. Rejestr SCON jest przeznaczony do programowego sterowania pracą portu. Rejestr SBUF jest rejestrem buforowym dla danych wysyłanych i odbieranych. W odróżnieniu od typowych rejestrów, umożliwiających zapamiętanie a później odczytanie tej samej informacji, rejestr SBUF składa się z dwu niezależnych rejestrów, umieszczonych pod tym samym adresem - rejestru nadajnika i odbiornika. Bufor nadajnika jest rejestrem wyłącznie do zapisu a bufor odbiornika - rejestrem wyłącznie do odczytu. Wysyłanie bajtu polega na zapisaniu jego wartości do bufora nadajnika (SBUF) - proces wysyłania danej, bit po bicie, jest wykonywany automatycznie. Zakończenie procesu wysyłania danej jest sygnalizowane ustawieniem bitu TI w rejestrze SCON. Odbieranie bitów danej jest również wykonywane w sposób automatyczny - po skompletowaniu bajtu danej jest on przekazywany do bufora odbiornika (SBUF) a zakończenie odbioru jest sygnalizowane ustawieniem bitu RI w rejestrze SCON. Niezależnie od trybu pracy, transmisja bitów danej jest zawsze rozpoczynana od bitu najmłodszego.



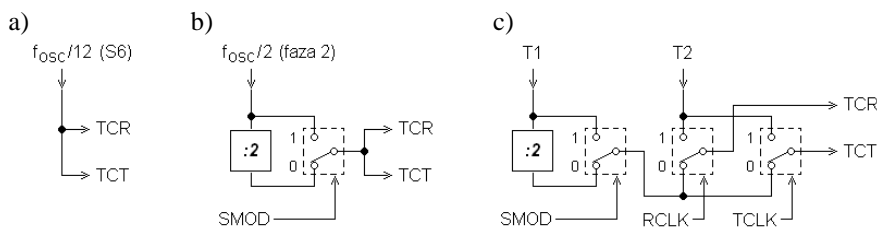
kolejność wysyłania bitów, niezależnie od trybu pracy portu szeregowego, jest zawsze rozpoczynana od bitu najmłodszego ..

Schemat blokowy portu transmisji szeregowej pokazano na rys.1.2.26. Port zawiera moduł kontrolera pracy, TRCB, dwa rejestry SBUF, rejestr przesuwany SIPO odbiornika, trzy przełączniki M1..M3, blok detektora stanu bitu BD oraz detektor opadającego zbocza sygnału odbieranego, TD. Elementami uzupełniającymi są dwa przerzutniki, D1 i D2. Praca portu jest definiowana stanem binarnym bitów SM0, SM1, SM2 oraz bitu REN - bity te są umieszczone w rejestrze SCON.

Wysyłanie szeregowo danej odbywa się za pośrednictwem rejestru przesuwanego typu PISO (ang. Parallel Input Serial Output). Sygnałem przesuwającym jest sygnał TSHIFT, generowany przez blok TRCB. W mikrokontrolerach MSC-51, rejestrem PISO jest rejestr SBUF nadajnika.



Rys. 1.2.26. Schemat blokowy portu transmisyjnego szeregowego¹.



Rys. 1.2.27. Źródła sygnału taktowania portu szeregowego: dla trybu 0(a) ; dla trybu 2 (b) oraz dla trybu 1 i 3 (c).

Odbieranie danych szeregowych odbywa się za pośrednictwem rejestru przesuwającego SIPO (ang. Serial Input Parallel Output). Sygnałem przesuwającym jest sygnał RSHIFT, generowany przez blok TRCB. Po skompletowaniu danych, stan rejestru SIPO jest przepisywany automatycznie do rejestru SBUF odbiornika.

Do wymiany informacji szeregowych z innym komputerem używa się 2 linii, RXD i TXD, które przypisano końcówkom mikrokontrolera, odpowiednio P3.0 i P3.1. Sygnał wprowadzany na końcówkę P3.1 (TXD) jest zawsze sygnałem wyjściowym mikrokontrolera.

¹ Układ portu transmisyjnego szeregowego, ze względu na swoją skomplikowaną budowę, jest pokazywany w literaturze za pomocą kilku rysunków - pokazany wyżej schemat blokowy jest próbą syntezy wielu schematów [1]

Dla trybu 1, 2 i 3, końcówka P3.0 (RXD) jest zawsze wejściem. Dla trybu 0, końcówka P3.0 może być wejściem lub wyjściem sygnału.

Tryb 0 - transmisja synchroniczna.

W trybie 0, port transmisji szeregowej mikrokontrolera może wymieniać informację z innymi urządzeniami w sposób synchroniczny. Wymiana informacji zawsze odbywa się porcjami 8-bitowymi. Sygnał taktowania nadajnika (TCT) i odbiornika (TCR) jest wspólny - jest nim faza S6 cyklu maszynowego (rys.1.2.3a). Szybkość wymiany informacji (szybkość transmisji) jest stała i wynosi $f_{osc}/12$.

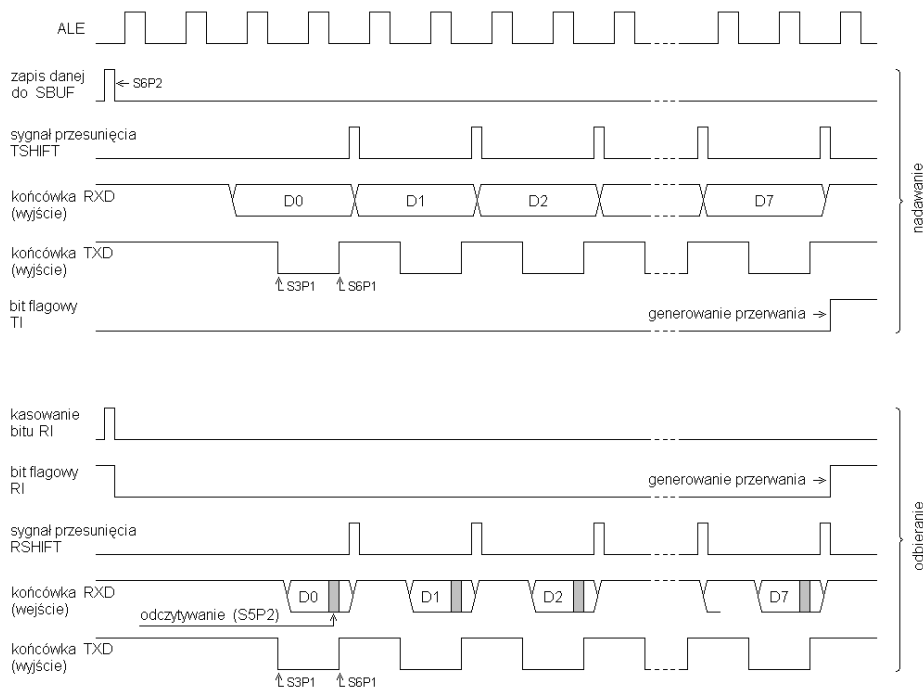
Sygnał zegarowy transmisji synchronicznej, SCLK, jest generowany przez blok kontrolny TRCB i wysyłany przez końcówkę P3.1 (TXD) - przełącznik M1 jest ustawiony w pozycję 0 (rys.1.2.26). Linia RXD, połączona z końcówką P3.0, pełni rolę linii danych. O tym, czy port szeregowy wysyła daną czy też ją odbiera, decyduje blok kontrolny TRCB za pośrednictwem przełącznika M2. Nadawanie jest możliwe, gdy do linii RXD jest dołączone wyjście SO rejestru nadajnika PISO (pozycja 1 przełącznika M2). Gdy przełączniki M2 i M3 znajdują się w pozycji 0, linia RXD jest dołączona bezpośrednio do wejścia SI rejestru SIPO - możliwe jest odbieranie danej. Dla trybu 0, przełączniki M1 i M3 są zawsze ustawione w pozycji 0.

Przebiegi czasowe sygnałów elektrycznych, charakterystyczne dla synchronicznego nadawania i odbierania danej, pokazano na rys.1.2.28. Wysyłanie i odbieranie informacji jest wewnętrznie synchronizowane poszczególnymi taktami i fazami sygnału zegarowego mikrokontrolera.

Nadawanie jest rozpoczynane po wpisaniu danej do rejestru SBUF. Po upływie 1 cyklu maszynowego, w fazie S6P2 tego cyklu, na linii RXD pojawia się najmłodszy bit danej, D0. Każdy następny cykl maszynowy, w fazie S6P2, powoduje pojawienie się sygnału przesunięcia TSHIFT, który powoduje przesuwanie bitów rejestrów SBUF w prawo - na linii RXD pojawiają się starsze bity danej. Równoległe do procesu wysyłania danej, zmieniany jest stan linii TXD: w fazach S3P1 cyklu maszynowego linia TXD przechodzi w stan 0 logicznego, a w fazach S6P1, linia TXD przechodzi w stan 1 logicznej. Proces nadawania kończy się ustawieniem bitu TI oraz linii TXD i RXD w stan jedynki logicznej - ustawiony bit TI może spowodować wygenerowanie przerwania. Bit TI jest umieszczony w rejestrze SCON.

Odbieranie jest rozpoczynane programowo przez wyzerowanie bitu RI, który jest umieszczony w rejestrze SCON. Przełącznik M2 jest przestawiany w pozycję 1 co powoduje dołączenie linii RXD do rejestru przesuwającego SIPO odbiornika. Identyfikacja jak przy nadawaniu, w tych samych momentach jest zmieniany stan linii TXD. W tych samych momentach, w których pojawiał się przy nadawaniu sygnał TSHIFT, teraz pojawia się sygnał RSHIFT, będący sygnałem przesunięcia dla rejestru SIPO. Przed przesunięciem, w fazie S5P2, jest do wejścia SI jest przepisywany stan linii RXD.

Każdy takt RSHIFT powoduje przesunięcie bitów rejestru w lewo a tym samym zapamiętanie informacji z wejścia SI. Po odebraniu 8 bitów danej, zawartość rejestru SIPO jest automatycznie przepisana do bufora SBUF odbiornika. W momencie przepisania, bit RI jest ustawiany w stan 1 logicznej co może spowodować wygenerowanie przerwania.



Rys. 1.2.28. Stan sygnałów przy nadawaniu i odbieraniu danych w trybie 0.

Przyjęcie przerwania spowodowane ustawieniem flag TI lub RI nie powoduje automatycznego, sprzętowego wyzerowania stanu tych wskaźników. Ustawione sprzętowo flagi muszą być zerowane w sposób programowy.

Tryb 1, 2 i 3 - transmisja asynchroniczna.

W trybie 1, 2 i 3, port transmisji szeregowej mikrokontrolera umożliwia wymianę informacji z innymi urządzeniami (komputerami) w sposób asynchroniczny. Poszczególne tryby różnią się pomiędzy sobą szybkością transmisji oraz liczbą przesyłanych bitów danej (tabela 1.2.9). W odróżnieniu od trybu synchronicznego, do wymiany informacji nie jest potrzebny sygnał synchronizacji. Dzięki temu, układ transmisji szeregowej mikrokontrolera ma do dyspozycji 2 końcówki, z których jedna jest sprzężona z wyjściem nadajnika (P3.1 - linia TXD) a druga jest połączona z wejściem odbiornika (P3.0 - linia RXD) - mikrokontroler może jednocześnie wysyłać i odbierać dane. Poprawność odbioru osiąga się poprzez taktowanie odbiornika sygnałem zegarowym, którego częstotliwość jest zgodna z częstotliwością taktowania nadajnika innego komputera

z dokładnością ok. $\pm 3\%$. W celu poinformowania odbiorcy o rozpoczęciu transmisji, przed blokiem bitów danej wysyła się bit startu, który zawsze jest zerem logicznym. Zakończenie wysyłania bloku bitów sygnalizuje się bitem stopu, który umieszcza się za bitami danej a jego wartość jest zawsze równa jedynce logicznej. W czasie, gdy dana nie jest wysyłana, linia sygnałowa nadajnika jest utrzymywana w stanie wysokim. Format przesyłanej danej trybu asynchronicznego pokazany jest na rys.1.2.29.



transmisja asynchroniczna jest rozpoczynana przez bit startu i kończona bitem stopu - bit startu zawsze jest zerem logicznym a bit stopu jedynką logiczną ..

Liczba bitów przesyłanej danej jest determinowana trybem pracy portu szeregowego. W trybie 1 jest transmitowanych 8 bitów danej - bity danej są pobierane z rejestru SBUF lub wpisywane do tego rejestru. W trybach 2 i 3 wysyła się lub odbiera 9 bitów danej. Pierwszych osiem bitów jest stanem rejestru SBUF. Dziewiąty bit danej (D8 z rys.1.2.29b) jest bitem ulokowanym w rejestrze SCON. W przypadku nadawania, dziewiątym bitem wysyłanej danej jest bit TB8. W przypadku odbioru, dziewiąty bit danej jest zapisywany w pozycję RB8 rejestru SCON.



Rys. 1.2.29. Format danej: 8 bitowej (a) i 9 bitowej (b).

Poszczególne tryby pracy różnią się pomiędzy sobą również źródłem sygnału taktowania liczników portu szeregowego, które umieszczone są w bloku kontrolnym TRCB. Źródła sygnału taktowania pokazano na rys.1.2.27b i c. W trybie 2, sygnały taktowania nadajnika (TCT) i odbiornika (TCR) są sygnałem wspólnym. Częstość taktowania jest ustawiana bitem SMOD (z rejestru PCON) i wynosi $f_{OSC}/2$ dla SMOD=1 oraz $f_{OSC}/4$ dla SMOD=0. W trybie 1 i 3, źródłem sygnału taktującego jest sygnał przepełnienia licznika T1 lub T2 a częstotliwość sygnału taktującego jest ustawiana przez stałą licznika. Dla mikrokontrolerów posiadających licznik T2, możliwe jest ustawienie różnych szybkości transmisji nadawania i odbioru. Ustawienie bitu SMOD (SMOD=1) pozwala na podwojenie szybkości transmisji w przypadku, gdy źródłem sygnału taktującego jest licznik T1.

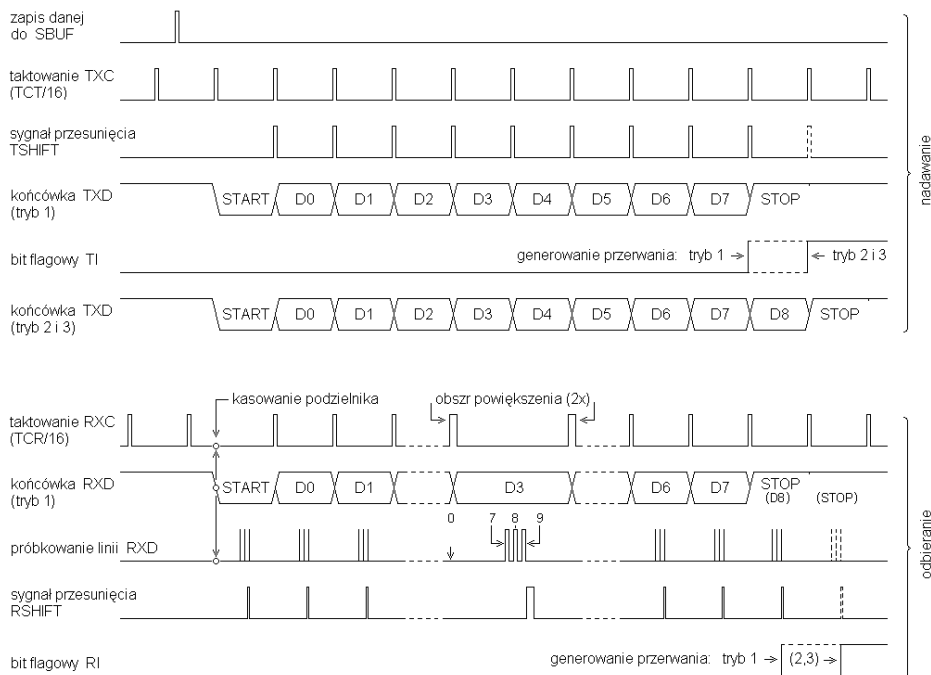
W przypadku transmisji asynchronicznej, ze względu na opisany dalej sposób detekcji sygnału odbieranego, do bloku kontrolnego TRCB doprowadza się sygnały taktowania, których częstotliwość musi być 16 razy większa od częstotliwości wysyłania lub odbierania bitów. W bloku TRCB następuje obniżenie częstotliwości tego sygnału do właściwej częstotliwości przez dwa, niezależne od siebie liczniki podzielnika, pracujące w trybie modulo 16 (liczniki 4-bitowe).

Podzielnik częstotliwości taktowania nadajnika generuje sygnał tak-towania TXC a podzielnik przeznaczony dla odbiornika, sygnał RXC. Dzięki obecności podzielników, w trybie 2, częstość pojawiania się poszczególnych bitów na wyjściu nadajnika wynosi $f_{OSC}/32$ dla SMOD=1 oraz $f_{OSC}/64$ dla SMOD=0. Ta sama uwaga dotyczy stałej liczników T1 lub T2 używanych do taktowania portu w trybie 1 i 3: liczniki muszą generować sygnał przepełnienia 16 razy częściej niż wynikałoby to z czasu określonego przez szybkość transmisji.



dla trybów 1..3, częstotliwość sygnału taktowania musi być 16 razy większa od częstotliwości wyznaczonej przez szybkość transmisji szeregowej ..

Pomijając różnice wynikające z liczby przesyłanych bitów danej, sprzętowy sposób wysyłania i odbierania bitów danej jest taki sam dla trybów 1, 2 i 3. Stan poszczególnych sygnałów portu szeregowego, powiązanych z nadawaniem i odbiorem danej, pokazano na rys.1.2.30. Pokazane na rysunku takty TXC oraz RXC wskazują miejsca, w których następuje przepełnienie 4-bitowych podzielników częstotliwości.



Rys. 1.2.30. Stan sygnałów przy nadawaniu i odbieraniu danej w trybie 1.. 3.

Nadawanie jest inicjowane przez wpisanie danej do rejestru SBUF. Dla trybu 2 lub 3, przed wpisaniem danej do rejestru SBUF, trzeba ustalić właściwy stan bitu TB8. Po zapisie danej do SBUF, w momencie pojawienia się pierwszego po zapisie taktu TXC, na wyjściu nadajnika pojawia się bit startu.

Następne takty TXC doprowadzają do wyjścia nadajnika kolejne bity danej z rejestru SBUF, rozpoczynając od bitu najmłodszego, D0. Po wysłaniu ostatniego bitu danej z rejestru SBUF (D7), w trybie 1 wysyłany jest bit stopu. W trybie 2 i 3, zamiast bitu stopu jest wysyłany bit TB8 - bit stopu jest wysyłany jako następny po TB8. W każdym przypadku, w momencie rozpoczęcia wysyłania bitu stopu, ustawiany jest bit flagowy stanu nadajnika, TI (TI=1). Stan 1 bitu TI może spowodować wygenerowanie przerwania w przypadku, gdy bit ES w rejestrze kontrolnym IE jest ustawiony (ES=1; przerwanie od portu transmisji szeregowej jest aktywowane - patrz opis IP, strona 55 lub 149).

Odbieranie danej jest realizowane za pośrednictwem rejestru przesuwonego SIPO oraz dwu detektorów: układu TD, będącego detektorem opadającego zbocza sygnału odbieranego oraz układu BD, który jest detektorem stanu odbieranego bitu (rys.1.2.26). Odbieranie jest inicjowane w momencie pojawienia się opadającego zbocza sygnału na linii RXD (P3.0) co może oznaczać pojawienie się na linii bitu startu. Po wykryciu tego zdarzenia, sygnał ST bloku TD powoduje wykasowanie wewnętrznego, 4-bitowego podzielnika częstotliwości taktowania odbiornika oraz wpisanie do rejestru SIPO wartości 1FFh. Po dokonaniu tej czynności, blok TD jest wyłączany. Obserwujący stan linii RXD detektor DB jest aktywowany w momencie, gdy licznik podzielnika generuje kolejne stany, 6, 7 i 8. Detektor DB testuje stan linii RXD w chwilach 6, 7 i 8 w ten sposób, że przepisuje na swoje wyjście Q stan, który wystąpił co najmniej 2 razy, kolejno po sobie. Taki sposób detekcji stanu linii RXD zapewnia odczyt stanu bitu w środku okresu przypisanego temu bitowi oraz eliminuje przypadkowe zmiany stanu sygnału, powstałe, np. w wyniku zakłóceń. W przypadku detekcji jedynek logicznej w bicie startu, co jest oznaką zaobserwowania zakłócenia a nie pojawienia się początku transmisji, układ TD jest aktywowany i wznowiane jest oczekiwanie na pojawienie się bitu startu. W przypadku detekcji zera, stan detektora BD jest wprowadzany do rejestru SIPO kolejnymi taktami RSHIFT. Próbkowanie stanu linii RXD jest identyczne jak w przypadku bitu startu. Po odebraniu wszystkich bitów danej i bitu stopu, stan rejestru SIPO jest przepisywany do rejestru SBUF odbiornika i bitu RB8. W momencie przepisywania, bit RI jest ustawiany w stan 1 logicznej co może spowodować wygenerowanie przerwania.

Szybkość transmisji.

Szybkość transmisji, w każdym przypadku, jest powiązana z częstotliwością własną generatora sygnału zegarowego mikrokontrolera, f_{osc} .

W trybie 0 szybkość jest stała i wynosi $f_{osc}/12$. W trybie 2 szybkość transmisji może przyjąć dwie wartości: $f_{osc}/32$ gdy bit SMOD=1 oraz $f_{osc}/64$ gdy bit SMOD=0. Bit SMOD znajduje się w rejestrze kontrolnym PCON (str.51). W trybie 1 lub 3 szybkość transmisji jest regulowana przez częstotliwość przepełniania się licznika T1. Licznik T1 może pracować w różnych trybach. Najczęściej, w typowych aplikacjach, licznik T1 pracuje jako czasomierz w trybie 2,

bez generacji przerwań. W tym trybie, licznik czasomierza T1 jest licznikiem 8-bitowym, który w momencie przepełnienia automatycznie pobiera nowy stan licznika z rejestru TH1 (patrz rozdział 1.2.5). Zawartość rejestru TH1 jest stałą licznika T1. Szybkość transmisji, w takim przypadku, jest określana wzorem:

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

Dla bardzo małych szybkości transmisji, pojemność licznika 8-bitowego jest niewystarczająca do wygenerowania sygnału taktowania portu szeregowego. W takim przypadku, można licznik T1 ustawić w tryb 1 (licznik 16-bitowy) z funkcją czasomierza. Szybkość transmisji, w takim przypadku, jest określana wzorem:

$$f_{\text{transm}} = \frac{2^{\text{SMOD}}}{32} * \frac{f_{\text{osc}}}{12 * [65536 - (256 * (\text{TH1}) + (\text{TL1}))]}$$

Jeżeli źródłem sygnału taktującego port szeregowy jest licznik T2, to szybkość transmisji jest określana wzorem:

$$f_{\text{transm}} = \frac{f_{\text{osc}}}{32 * [65536 - (256 * (\text{RCAPH}) + (\text{RCAPL}))]}$$

Tabela 1.2.10. Szybkość transmisji portu szeregowego, pracującego w trybie 1 lub 3, w funkcji częstotliwości własnej generatora sygnału zegarowego mikrokontrolera oraz stałej przeładowania licznika T1.

szybkość transmisji [bit/s]	f_{osc} [MHz]	SMOD	częstość taktowania [kHz]	stała licznika [hex]	odchylenie [%]
57600	11,0592	1	921,6	FF	0
38400	11,0592	1	921,6	E8	0
14400	11,0592	1	921,6	C0	0
19200	11,0592	1	921,6	FD	0
9600	11,0592	0	921,6	FD	0
4800	11,0592	0	921,6	FA	0
2400	11,0592	0	921,6	F4	0
1200	11,0592	0	921,6	E8	0
9600	12,0	0	1000,0	FD	+8,5
4800	12,0	0	1000,0	F9	-7,0
2400	12,0	0	1000,0	F3	+1,6
1200	12,0	0	1000,0	E6	+1,6

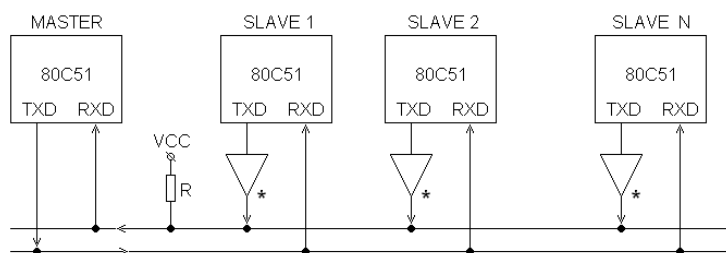
W tabeli 1.2.10 przedstawiono kilka wartości stałej licznika T1 w funkcji szybkości transmisji, charakterystycznych dla standardu RS232. Układ licznikowy T1 pracuje jako czasomierz w trybie 2. W tabeli pokazano odchylenie szybkości transmisji od wartości zadanej.

Przy zastosowaniu rezonatora kwarcowego o częstotliwości własnej 12MHz, zgodność szybkości transmisji ze standardem RS232 jest osiągnięta dla szybkości mniejszej od 4800 b/s. Dla rezonatora, o częstotliwości własnej 11,0592MHz, zgodność szybkości uzyskiwanej z zadaną jest pełna (100%).

Komunikacja w systemie wieloprocesorowym.

Rejestr kontrolny portu szeregowego, SCON, posiada 2 bity, których stan wpływa na odbiór danej. Bit REN włącza lub wyłącza odbiornik. Gdy REN=0 to odbiornik portu jest wyłączony. Gdy REN=1 to odbiornik jest włączony i może wprowadzać odebraną daną do rejestru SBUF. O tym, czy odebrana dana będzie przepisana do rejestru SBUF czy też nie, decyduje stan bitu SM2, który nazywany jest bitem maskowania odbioru. Jeżeli SM2=0 to każda odebrana dana jest przepisywana do SBUF w każdym przypadku (i do bitu RB8 dla trybu 2 i 3). Gdy SM2=1, to przepisanie odebranej danej do rejestru SBUF i ewentualne wygenerowanie przerwania jest możliwe wyłącznie w przypadku, gdy dziewiąty bit odebranej danej jest jedynką logiczną. Dla trybu 1 ten warunek jest spełniany automatycznie w przypadku poprawnego odbioru - 9 bit jest bitem stopu i powinien być jedynką logiczną. Gdy dana jest odbierana w trybie 2 lub 3, to nadawca decyduje, czy dana będzie odebrana czy też nie.

Właściwość maskowania odbioru może być wykorzystana do stworzenia prostego protokołu komunikacyjnego w systemie wieloprocesorowym pomiędzy mikrokontrolerem głównym a mikrokontrolerem podrzędnym (ang. master and slave) lub grupą mikrokontrolerów podrzędnych. Schemat blokowy takiego połączenia pokazano na rys.1.2.31. Linia TXD mikrokontrolera głównego jest dołączona do linii RXD mikrokontrolerów podrzędnych. Linie TXD mikrokontrolerów podrzędnych, za pośrednictwem bramek buforowych z otwartym kolektorem są połączone z wejściem RXD mikrokontrolera głównego. Opornik R jest opornikiem polaryzującym.



Rys. 1.2.31. Komunikacja w systemie wieloprocesorowym.

Zakładając, że:

- każdy mikrokontroler podrzędny posiada indywidualny numer i ma włączone maskowanie odbioru;
- przesyłanie adresu jest powiązane z przesłaniem 9 bitu o wartości 1 - adres może być wysyłany wyłącznie przez mikrokontroler główny;

- przesyłanie danej jest powiązane z przesłaniem 9 bitu o wartości 0 - dana może być wysyłana przez mikrokontroler główny oraz każdy, wskazany adresem mikrokontroler podrzędny;

można zbudować protokół wymiany informacji wg następujących zasad:

- mikrokontroler główny wysyła adres do wszystkich urządzeń podrzędnych - w bajcie adresowym mogą się mieścić dodatkowe informacje o sposobie prowadzenia wymiany danych;
- wskazany adresem mikrokontroler podrzędny wyłącza maskowanie odbioru i od tego momentu może odbierać każdą daną od mikrokontrolera głównego, której 9 bit jest zerem; może również wysyłać dane z wyzerowanym 9 bitem do mikrokontrolera głównego, wg wcześniejszej dyspozycji;
- po przekazaniu wszystkich danych lub po odbiorze innego adresu, wskazany poprzednio mikrokontroler włącza maskowanie odbioru i przechodzi w stan oczekiwania na ponowną aktywację.

Oczywiście protokół wymiany informacji w systemie wieloprocesorowym można zrealizować bez funkcji maskowania odbioru ale wtedy każdy mikrokontroler podrzędny musiałby odbierać wszystkie elementy transmisji i prowadzić ich analizę - a to mogłoby być procesem czasochłonnym.

Rejestry i bity kontrolne portu szeregowego.

Jak już wspomniano, do sterowania pracą portu transmisji szeregowej służą bity kontrolne, zgrupowane w rejestrze SCON. Rejestr SCON może być adresowany bitowo. Do nadawania i odbioru danej jest przeznaczony rejestr SBUF, który umieszczono w strefie SFR pod adresem 99h.

SCON:	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	98h
	9Fh	9Eh	9Dh	9Ch	9Bh	9Ah	99h	98h	

Rys. 1.2.32. Rejestr SCON.

Działanie poszczególnych bitów rejestru jest następujące:

SM0,SM1 - ustawianie trybu pracy portu szeregowego (patrz tabela 1.2.9):

- 00 - tryb 0;
- 01 - tryb 1;
- 10 - tryb 2;
- 11 - tryb 3;

SM2 - maskowanie odbioru:

- 0 - bit RI jest zawsze ustawiany po odebraniu danej; ta nastawa jest wymagana dla trybu 0 (SM2 = 0);
- 1 - bit RI jest ustawiany gdy:
 - w trybie 1 odebrano bit stopu;
 - w trybie 2 lub 3 dziewiąty bit danej jest jedyneką (RB8 = 1);

- REN** - włączanie lub wyłączanie odbiornika:
 0 - odbiornik wyłączony;
 1 - odbiornik włączony;
- TB8** - 9 bit wysyłanej danej;
- RB8** - 9 bit odebranej danej - w trybie 1 wpisywany jest tu bit stopu (RB8 = 1); w trybie 0 bit jest nieużywany;
- TI** - bit flagowy stanu nadawania - może spowodować zgłoszenie przerwania; bit jest ustawiany sprzętowo w momencie zakończenia wysyłania danej; kasowanie bitu wyłącznie programowe;
- RI** - bit flagowy stanu odbioru - może spowodować zgłoszenie przerwania; bit jest ustawiany sprzętowo w momencie zakończenia odbioru danej; kasowanie bitu wyłącznie programowe.

W przypadku gdy port transmisji szeregowej jest taktowany przez licznik T1, możliwe jest zwiększenie szybkości transmisji przez ustawienie bitu SMOD, który znajduje się w rejestrze PCON (po uruchomieniu mikrokontrolera, bit SMOD jest wyzerowany). Rejestr PCON nie może być adresowany bitowo.

PCON:	SMOD	-	-	-	GF1	GF2	PD	IDL	87h
-------	------	---	---	---	-----	-----	----	-----	-----

Rys. 1.2.33. Rejestr PCON.

Działanie poszczególnych bitów rejestru jest następujące:

- SMOD** - bit mnożnika szybkości transmisji szeregowej:
 0 - częstotliwość sygnału z czasomierza T1 jest dzielona przez 2;
 1 - sygnał z T1 bezpośrednio taktuje port szeregowy;
- GF0,GF1** - bity ogólnego przeznaczenia (tylko układy CHMOS);
- PD** - bit aktywacji stanu obniżonego poboru mocy;
- IDL** - bit aktywacji pracy w trybie jałowym;

Bity GF0 i GF1 mogą być wykorzystane w dowolny sposób. Bity PD i IDL są wykorzystywane w trybie pracy z obniżonym poborem mocy.

1.2.7. Kontroler przerwania.

Jak już wspomniano, pojawienie się aktywnego stanu sygnału przerwania od urządzenia I/O powoduje przerwanie bieżąco wykonywanego programu i rozpoczęcie wykonywania innego, nazywanego programem obsługi przerwania. W przypadku istnienia wielu źródeł przerwania w systemie mikroprocesorowym niezbędnym staje się odpowiednie zarządzanie porządkiem wykonywania przerwania. Rola tę spełnia układ sterowania przerwaniem - kontroler przerwania.

Zadaniem kontrolera jest odebranie informacji o przerwaniach, dokonanie arbitrażu ważności zgłoszonych przerwania oraz przekazanie mikroprocesorowi informacji o tym, że przerwanie pojawiło się i jak należy je wykonać (wskazanie

numeru procedury przerwaniowej). Rozstrzygnięcie tych, wydawałoby się, prostych problemów nie jest banalną sprawą. Kontroler musi "śledzić" działanie mikroprocesora. Przykładowo, po zgłoszeniu przerwania do mikroprocesora powinien on zapamiętać priorytet tego przerwania i testować, czy mikroprocesor zakończył już wykonywanie programu obsługi przerwania czy też nie. Jest to ważne z tego powodu, że w przypadku, gdy mikroprocesor wykonuje program obsługi przerwania a do kontrolera zostanie zgłoszone inne, to musi on sprawdzić priorytet nowego przerwania i zdecydować co robić dalej. W przypadku, gdy mikroprocesor obsługuje wcześniej zgłoszone przerwania a nowe przerwania ma wyższy priorytet od obsługiwanego, kontroler przerwania musi przekazać mikroprocesorowi informacje o nowym przerwaniu. W przypadku, gdy mikroprocesor obsługuje przerwania o priorytecie wyższym od nowo zgłoszonego, kontroler musi zapamiętać fakt nadejścia nowego zgłoszenia oraz jego priorytet. Natychmiast po zakończeniu wykonywania przez mikroprocesor obsługi wcześniej zgłoszonego przerwania, kontroler musi zgłosić mikroprocesorowi fakt pojawienia się nowego. Ponieważ kontroler musi działać w sposób natychmiastowy i automatyczny, jego konstrukcja sprzętowa może być bardzo skomplikowana.

W strukturze mikrokontrolera 80C51 wbudowano 3 urządzenia I/O: liczniki T0 i T1 oraz port transmisji szeregowej. W mikrokontrolerze 80C52 jest dodatkowy licznik T2 - układ posiada zatem 4 urządzenia I/O. Każde z urządzeń I/O jest w stanie wygenerować sygnał przerwania. W mikrokontrolerze przewidziano również możliwość zgłaszania przerwania z urządzeń zewnętrznych - do wprowadzenia tych sygnałów zarezerwowano 2 końcówki mikrokontrolera: INT0 oraz INT1. Kontroler przerwania musi zatem obsłużyć 5 źródeł przerwania w przypadku układu 80C51 i 6 w przypadku 80C52.

W układach rodziny MCS-51, każdemu przerwaniu przypisano dedykowany wektor przerwania (adres początkowy procedury obsługi przerwania) oraz sztywny priorytet wykonywania przerwania. Taki model można nazwać modelem statycznym. Przyjęto, że adresy początku procedur przerwaniowych będą umiejscowione w pamięci programu wg kolejności podanej w tabeli 1.2.11. Powyższy porządek definiuje również priorytet przerwania: najwyższy priorytet posiada przerwania wygenerowane przez urządzenie zewnętrzne, INT0, a najniższy, przerwania wygenerowane przez licznik T2. W tabeli oznaczono priorytet przerwania cyframi - cyfrze 0 przypisany jest najwyższy priorytet.

Urządzenia mogą zgłaszać przerwania poprzez przypisanie stanu jedynek logicznej do właściwych tym urządzeniom bitów kontrolnych - flag wystąpienia przerwania. Sam fakt ustawienia flagi przerwania nie wystarcza do wywołania procedury obsługi przerwania. Do tego celu jest potrzebne tzw. aktywowanie przerwania. Bity decydujące o tym, czy przerwania będzie wykonane czy też nie, są zgrupowane w rejestrze kontrolnym **IE** (ang. Interrupt Enable Register). Rejestr IE jest często nazywany maską przerwania. Po włączeniu mikrokontrolera lub skasowaniu sygnałem kasowania, wszystkie bity rejestru są wyzerowane -

nie jest możliwe wykonywanie jakichkolwiek przerw. Bity trzeba ustawić programowo.

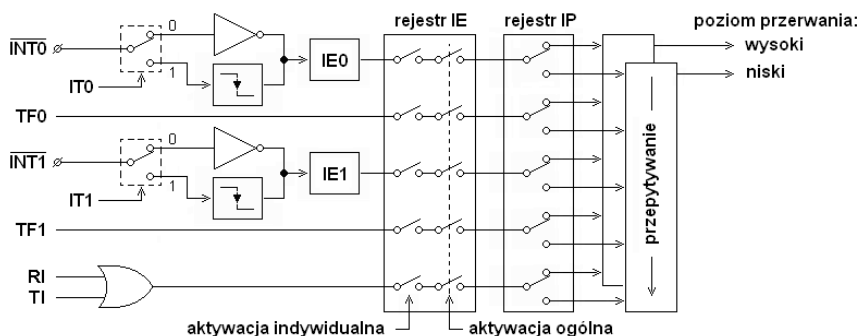
Tabela 1.2.11. Adresy wektorów przerw dla 80C51/52.

priorytet	adres wektora przerw	opis
0	03h	przerwanie od urządzenia zewnętrznego - z wejścia INT0;
1	0Bh	przerwanie od licznika T0;
2	13h	przerwanie od urządzenia zewnętrznego - z wejścia INT1;
3	1Bh	przerwanie od licznika T1;
4	23h	przerwanie od portu transmisji szeregowej;
5	2Bh	przerwanie od licznika T2 (dla 80C52)

Priorytet przerw zastosowany w układach rodziny MCS-51 dość dobrze oddaje statystyczny rozkład występowania priorytetu w typowych systemach mikroprocesorowych. Ponieważ jednak nie wszystkie systemy są "typowe", konstruktorzy mikrokontrolerów MCS-51 przewidzieli możliwość pewnej modyfikacji systemu przerw poprzez dodanie dodatkowego elementu, nazywanego *poziomem priorytetu*. Mówimy w tym przypadku o dwupoziomym systemie przerw. Jeżeli nazwać opisany do tej pory sposób wykonywania przerw poziomu podstawowym to dodatkowy element jest poziomem wysokim. Priorytet wykonywania przerw na poziomie wysokim jest identyczny jak dla poziomu podstawowego - przerwania są wykonywane wg kolejności podanej w tabeli 1.2.11. Wykonywanie przerw w układzie dwupoziomym jest realizowane w ten sposób, że pierwszeństwo wykonywania jest zawsze przypisane zgłoszeniom, które przypisano do wyższego poziomu.

O tym, które z przerw znajdzie się na poziomie wyższym, decyduje stan bitów zgrupowanych w rejestrze kontrolnym **IP** (ang. Interrupt Priority Register). Po włączeniu mikrokontrolera lub skasowaniu sygnałem kasowania, wszystkie bity rejestru są wyzerowane - wszystkie przerwania są wywoływane z poziomu podstawowego. W celu podwyższenia priorytetu wybranego przerwania, odpowiadający mu bit musi być ustawiony programowo - bit o wartości 1 wskazuje na podwyższenie priorytetu.

Schemat blokowy układu przerw pokazano na rysunku 1.2.34. Patrząc od lewej strony, na rysunku pokazano kolejno: źródła przerw, rejestr IE, rejestr IP oraz elementy poziomu podstawowego i wysokiego, generujących wspólnie wektor przerw. Z punktu widzenia transmisji sygnałów, rejestr IE jest zespołem styczników a rejestr IP jest zespołem przełączników, który kieruje sygnały przerw do poziomu podstawowego lub wysokiego.



Rys. 1.2.34. Schemat blokowy kontrolera przerwań w 80C51.

Tabela 1.2.12. Priorytet przerwań w systemie 2 poziomowym.

poziom	priorytet formalny	priorytet rzeczywisty	adres wektora przerwań	opis
wysoki	0		03h	przerwanie od urządzenia zewnętrznego - z wejścia INT0;
	1	0	0Bh	przerwanie od licznika T0;
	2		13h	przerwanie od urządzenia zewnętrznego - z wejścia INT1;
	3		1Bh	przerwanie od licznika T1;
	4	1	23h	przerwanie od portu transmisji szeregowej;
	5		2Bh	przerwanie od licznika T2 (dla 80C52)
podstawowy	0	2	03h	przerwanie od urządzenia zewnętrznego - z wejścia INT0;
	1		0Bh	przerwanie od licznika T0;
	2	3	13h	przerwanie od urządzenia zewnętrznego - z wejścia INT1;
	3	4	1Bh	przerwanie od licznika T1;
	4		23h	przerwanie od portu transmisji szeregowej;
	5	5	2Bh	przerwanie od licznika T2 (dla 80C52)

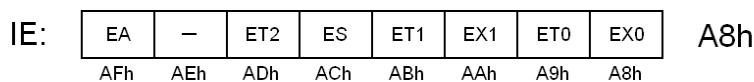
Jak zrozumieć działanie dwupoziomowego systemu przerwań?

W tabeli 1.2.12 pokazano rozmieszczenie przerwań na poziomie wysokim i podstawowym. Priorytet przerwań na każdym z tych poziomów jest taki sam i został przedstawiony w tabeli jako priorytet formalny. Dzięki obecności przełączników w systemie przerwań, które sterowane są przez rejestr IP, sygnały zgłoszenia przerwania pojawią się albo na poziomie podstawowym albo wysokim.

Założmy, dla przykładu, że z punktu widzenia systemu mikroprocesorowego, bardzo ważnym zdarzeniem będzie przepełnienie licznika T0 oraz pojawienie się danej, odebranej przez port transmisji szeregowej. Założmy ponadto, że aktywne będą wszystkie źródła przerw. Właściwe ustawienie bitów rejestru IP spowoduje skierowanie sygnałów przerw od licznika T0 i portu transmisji szeregowej na wyższy poziom. W wyniku tego przekierowania powstanie nowy obraz priorytetu przerw pokazany w tabeli 1.2.12 i podkreślony pogrubioną czcionką. Przerwanie od licznika T0 i portu transmisji szeregowej zostały przesunięte na pozycje czołowe. Należy przy tym zauważyć, że pomimo tak dokonanego przesunięcia priorytetu, w dalszym ciągu przerwanie od licznika T0 ma wyższy priorytet niż przerwanie od portu transmisji szeregowej. W dwupoziomowym systemie przerw istnieje możliwość przegrupowania priorytetu przerw ale ma ona swoje ograniczenia - na poziomie wysokim, w dalszym ciągu jest zachowany priorytet poziomu podstawowego.

Rejestry i bity sterujące układu kontrolera przerw.

Jak już wspomniano, każde źródło przerwania może być indywidualnie blokowane lub aktywowane przez odpowiednie ustawienie bitów w rejestrze kontrolnym sterownika przerw, IE. Rejestr IE może być adresowany bitowo.



Rys. 1.2.35. Rejestr kontrolny IE.

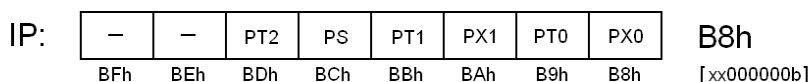
Wszystkim bitom rejestru przypisano identyczny sposób działania:

- 0 - obsługa przerwania zablokowana;
- 1 - obsługa przerwania aktywna;

Znaczenie poszczególnych bitów jest następujące:

- EX0** - bit aktywacji przerwania zewnętrznego INT0;
- ET0** - bit aktywacji przerwania zewnętrznego licznika T0;
- EX1** - bit aktywacji przerwania zewnętrznego INT1;
- ET1** - bit aktywacji przerwania zewnętrznego licznika T1;
- ES** - bit aktywacji przerwania portu transmisji szeregowej;
- ET2** - bit aktywacji przerwania zewnętrznego licznika T2;
- EA** - bit aktywacji wszystkich przerw mikrokontrolera.

O pozostawieniu przerwania na poziomie podstawowym lub przesunięciu go na poziom wyższy decyduje stan bitów umieszczonych w rejestrze IP. Rejestr IP może być adresowany bitowo.



Rys. 1.2.36. Rejestr kontrolny IP.

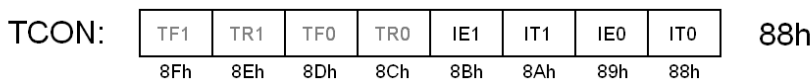
Wszystkim bitom rejestru przypisano identyczny sposób działania - stan bitu definiuje wybór poziomu priorytetu przerwania:

- 0 - poziom podstawowy;
- 1 - poziom wysoki;

Znaczenie poszczególnych bitów jest następujące:

- PX0** - bit aktywacji przerwania zewnętrznego INT0;
- PT0** - bit aktywacji przerwania zewnętrznego licznika T0;
- PX1** - bit aktywacji przerwania zewnętrznego INT1;
- PT1** - bit aktywacji przerwania zewnętrznego licznika T1;
- PS** - bit aktywacji przerwania portu transmisji szeregowej;
- PT2** - bit aktywacji przerwania zewnętrznego licznika T2;

Przyglądając się schematowi blokowemu kontrolera przerw z rys. 1.2.34, można zauważyć, że bitami flagowymi zgłoszenia przerw zewnętrznych, INT0 i INT1, są bity EI0 oraz EI1. O sposobie ich ustawiania decyduje stan przełączników, które sterowane są stanem bitów IT0 oraz IT1. Bity te są umieszczone w rejestrze TCON. Działanie bitów TF1, TR1, TF0 i TR0, powiązanych z licznikami T0 i T1, opisano już w rozdziale 1.2.5 (podrozdział "Bity kontrolne liczników T0 i T1").



Rys. 1.2.37. Rejestr kontrolny TCON.

Rejestr TCON może być adresowany bitowo. Działanie poszczególnych bitów kontrolnych, które powiązane są z przerwaniami zewnętrznymi INT0 oraz INT1, można przedstawić następująco:

- IT0** - ustawianie sposobu zgłoszenia przerwania zewnętrznego INT0:
 - 0 - zgłoszenie poziomem niskim sygnału;
 - 1 - zgłoszenie zboczem opadającym sygnału;
- IE0** - flaga zgłoszenia przerwania INT0:
 - 0 - stan pasywny - nic się nie dzieje;
 - 1 - stan aktywny - wykryto poziom niski sygnału lub jego opadające zbocze;
- IT1** - ustawianie sposobu zgłoszenia przerwania zewnętrznego INT1:
 - 0 - zgłoszenie poziomem niskim sygnału;
 - 1 - zgłoszenie zboczem opadającym sygnału;
- IE1** - flaga zgłoszenia przerwania INT1:
 - 0 - stan pasywny - nic się nie dzieje;
 - 1 - stan aktywny - wykryto poziom niski sygnału lub jego opadające zbocze;

Wykrycie zmiany stanu sygnału zewnętrznego, z wartości 1 na 0 (wykrycie zbocza opadającego sygnału), jest zdarzeniem o bardzo krótkim czasie trwania - teoretycznie nieskończenie małym. W wyniku tego zdarzenia jest ustawiana flaga zgłoszenia przerwania, IE. Z punktu widzenia mikroprocesora, został ustawiony bit konkretnego rejestru i stan tego bitu może być w każdej chwili zmieniony, np. przez strukturę sprzętową mikrokontrolera.

W przypadku zadeklarowania sposobu zgłaszania przerwania przez poziom końcówki INT, stan bitu IE jest ściśle powiązany ze stanem sygnału zewnętrznego i jest jego negacją (patrz rysunek 1.2.34). Struktura wewnętrzna mikrokontrolera nie ma żadnego, bezpośredniego wpływu na stan tego bitu.

Po zaobserwowaniu stanu wysokiego bitu IE rozpoczynane jest wykonywanie programu obsługi przerwania. Ponieważ program obsługi przerwania jest takim samym programem jak każdy inny, to w końcowej fazie każdego cyklu rozkazowego jest testowany stan flag przerwania. W przypadku znalezienia zgłoszenia z priorytetem wyższym od obecnie obsługiwanego, mikroprocesor może rozpocząć wykonywanie innego programu obsługi. W przypadku jednak, gdy zaobserwowane zgłoszenie ma priorytet równy lub niższy od obecnie obsługiwanego, przyjęcie tego przerwania nastąpi dopiero po zakończeniu bieżąco wykonywanego programu obsługi przerwania.

W każdym przypadku, bit IE musi być wyzerowany przed zakończeniem obsługi przerwania. Zaniedbanie tej czynności doprowadziłoby do ponownego przyjęcia tego samego przerwania. W przypadku zgłaszania przerwania za pośrednictwem zbocza opadającego, w momencie przyjęcia przerwania, bit IE jest zerowany w sposób automatyczny. W przypadku zgłaszania przerwania niskim poziomem, w ramach programu obsługi przerwania, do urządzenia zewnętrznego musi być wysłane żądanie usunięcia stanu aktywnego sygnału przerwania - końcówka INT musi być ustawiona w stan wysoki. A tę czynność może wykonać wyłącznie urządzenie zewnętrzne.

1.2.8. Dodatkowe urządzenia I/O mikrokontrolera 89S8253.

Mikrokontroler AT89S8253 jest współczesnym odpowiednikiem układu 80C52, który w stosunku do swojego pierwowzoru oferuje kilka dodatkowych możliwości, między innymi:

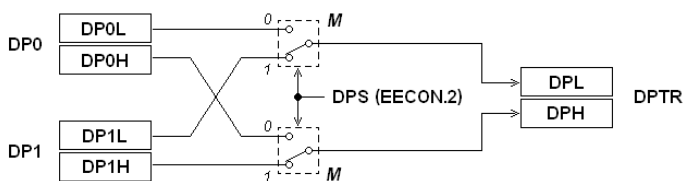
- 2kB pamięci danych EEPROM;
- łącze szeregowe SPI (ang. Serial Peripheral Interface);
- programowalny układ autoalarmu (ang. Watchdog Timer, WDT)
- podwójny wskaźnik danych DPTR;
- czteropoziomowy kontroler przerwania;
- układ monitora napięcia zasilania (ang. Internal Power-on Reset)
- opcjonalny tryb szybkości pracy (x2).

W niniejszym opracowaniu, ze względu na zaproponowany zestaw ćwiczeń laboratoryjnych, zostaną omówione: układ podwójnego rejestru DPTR i układ

autoalarmu WDT. Pozostałe funkcje mikrokontrolera mogą być użyte w ramach zajęć indywidualnych z zastrzeżeniem, że nie można w systemie FTSM_51 (opis systemu na stronie 65) stosować trybu zwiększonej szybkości pracy (x2).

Wskaźniki DPTR.

W rdzeniu mikrokontrolera 80C51/52 używany jest pojedynczy wskaźnik położenia danych, DPTR. Wykonywanie procedur kopiowania bloków danych przy pomocy tego rejestru jest czynnością dość uciążliwą - bez przerwy trzeba wpisywać do niego kolejne adresy źródła danej i adresy miejsca przeznaczenia. Obecność dwóch wskaźników bardzo ułatwiłaby proces kopiowania - jeden wskaźnik pokazywałby adres źródła danych a drugi, adres miejsca przeznaczenia.



Rys. 1.2.38. Układ podwójnego wskaźnika danych, DPTR.

Układ 89S8253 oferuje 2 wskaźniki DPTR, które nazwano DP0 i DP1. Każdy z nich składa się z dwu rejestrów, DPxL i DPxH, oznaczonych jako DP0L i DP0H oraz DP1L i DP1H. Rejestry są umieszczone w strefie SFR pod kolejnymi adresami (zaczynając od DP0L): 82h, 83h, 84h i 85h. Z punktu widzenia instrukcji programu, używających rejestru DPTR, rejestr ten jest zawsze pojedynczym rejestrem. O tym, który rejestr jest w danym momencie używany, decyduje stan bitu DPS, który jest umieszczony w rejestrze kontrolnym EECON, pod adresem 96h strefy SFR. Sposób wybierania rejestru DP0 lub DP1 pokazano na rysunku 1.2.38 - wybór jest dokonywany za pośrednictwem 2 jednobitowych multiplekserów M, sterowanych bitem DPS. Gdy bit DPS jest zerem to rejestrem DPTR jest rejestr DP0; gdy bit DPS jest jedynką to rejestrem DPTR jest rejestr DP1.

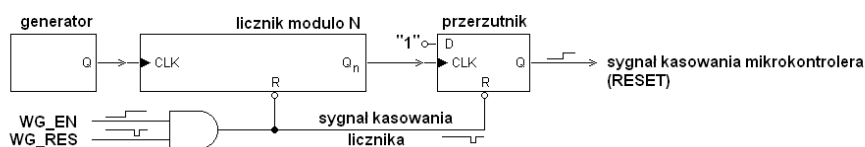


Rys. 1.2.39. Rejestr kontrolny EECON (obraz uproszczony do bitu DPS).

Pokazany na rysunku 1.2.39 rejestr EECON nie może być adresowany bitowo. W rejestrze nie pokazano bitów sterujących procesem zarządzania pamięcią EEPROM - bity te zaznaczono gwiazdkami (dostęp do EEPROM nie jest zadaniem laboratoryjnym).

Układ WDT.

Bardzo często zdarza się, że oparte o systemy mikroprocesorowe, przemysłowe urządzenia sterujące pracują w środowisku o dużym poziomie zakłóceń elektromagnetycznych. W takim środowisku może zdarzyć się, że system mikroprocesorowy (sterownik mikroprocesorowy) zawiesi się. Zawieszenie pracy może nastąpić, np. w wyniku złej interpretacji kodu instrukcji i rozpoczęcia wykonywania innej części programu, nie związanej z bezpośrednią obserwacją stanu środowiska systemu mikroprocesorowego. Jedynym środkiem zaradczym, w takim przypadku, jest wyłączenie i ponowne uruchomienie zablokowanego systemu. Powinno to nastąpić jak najszybciej po zaistnieniu awarii a ponowne uruchomienie sterownika nie powinno zaburzyć cyklu technologicznego, w którym ten sterownik pracował. Ponowny restart sterownika można osiągnąć przez zastosowanie układu czuwającego (alarmowego), który w dalszej części opracowania będzie opisywany skrótem WDT (ang. watchdog timer). Układ ten, w chwili obecnej, jest wbudowywany w prawie każdy nowy model mikrokontrolera.

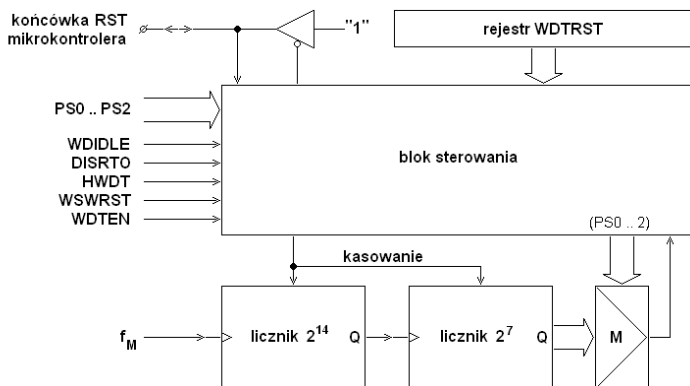


Rys. 1.2.40. Uproszczony schemat blokowy układu alarmowego "watchdog".

Konstrukcja układu alarmowego może wyglądać tak, jak to przedstawiono na rysunku 1.2.40. Układ składa się z generatora, licznika oraz przerzutnika D. Licznik i przerzutnik D są kasowane sygnałem wyjściowym bramki AND. W przypadku, gdy sygnał kasowania licznika przejdzie w stan jedynki logicznej, licznik zlicza takty generatora a po swoim przepełnieniu (po zarejestrowaniu N taktów sygnału generatora), zmienia stan przerzutnika D - generowany jest sygnał kasowania mikrokontrolera. W celu niedopuszczenia do wygenerowania tego sygnału, układ licznika musi być kasowany przed momentem swojego przepełnienia. Do tego celu może służyć sygnał WG_RES, który powinien być wytwarzany, np. przez dedykowany podprogram, okresowo aktywowany w obszarze pętli programowej. Pokazany na rysunku sygnał WG_EN służy do aktywowania układu alarmowego - sygnał musi być sprzętowo zerowany w momencie rozpoczynania pracy przez mikrokontroler i powinien być ustawiany przez program w przypadku chęci użycia układu alarmowego.

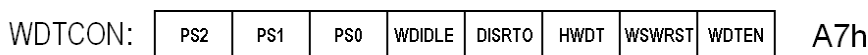
Mikrokontroler 89S8253 jest wyposażony w układ WDT. Uproszczony schemat tego układu pokazano na rysunku 1.2.41. Do swojej pracy, układ WDT wykorzystuje licznik binarny, zliczający w górę i taktowany z częstotliwością wykonywania cykli maszynowych, f_M ($f_{OSC}/12$). Licznik jest zbudowany z 21 elementów typu T. Wyjściem licznika jest, wyselekcjonowane przez multiplexer M, jedno z siedmiu najstarszych wyjść licznika. Dzięki możliwości nastawy multiplexera bitami PS0, PS1 i PS2 z rejestru WDTCON, otrzymuje się licznik

o regulowanej wartości trybu modulo - pojemność licznika jest regulowana w zakresie od 2^{14} do 2^{21} (patrz tabela 1.2.13). Oznacza to, że w przypadku taktowania mikrokontrolera z częstotliwością 12MHz, czas potrzebny na zapelnienie licznika można regulować w zakresie od 16ms do ponad 2s.



Rys. 1.2.41. Uproszczony schemat blokowy układu alarmowego "watchdog" mikrokontrolera 89S8253.

Praca układu WDT jest sterowana przez dwa rejestry strefy SFR: rejestr WDTCON (adres: 0A7h) oraz rejestr pomocniczy WDTRST (adres: 0A6h).



Rys. 1.2.42. Rejestr kontrolny WDTCON.

Rejestr WDTCON nie może być adresowany bitowo. Działanie poszczególnych bitów kontrolnych rejestru można przedstawić następująco:

- PS0..2** - bity ustawiające pojemność licznika WDT (patrz tabela 1.2.13);
- WDIDLE** - ustawianie sposobu pracy WDT w trybie pracy jałowej:
- 0 - stan aktywny - układ pracuje;
 - 1 - stan pasywny - praca układu jest blokowana;
- DISRTO** - ustawianie sposobu sterowania stanem końcówki RST mikrokontrolera:
- 0 - końcówka RST jest wprowadzana w stan jedynek logicznej przez okres 96 taktów zegara (pracuje jako wyjście przez okres 8 cykli maszynowych);
 - 1 - końcówka RST jest wyłącznie w stanie wejściowym;
- HWDT** - ustawianie trybu sterowania pracą WDT:
- 0 - tryb sterowania programowego;
 - 1 - tryb sterowania sprzętowego;

- WSWRST** - bit programowego kasowania układu licznikowego WDT:
 0 - stan wymuszany sprzętowo po skasowaniu licznika;
 1 - zlecenie skasowania stanu licznika;
- WDTEN** - bit programowego aktywowania układu WDT:
 0 - zablokowane układ licznikowy;
 1 - układ licznikowy odblokowany;

Tabela 1.2.13. Pojemność licznika WDT i czas zliczania w funkcji nastawy PS0.. PS2 (dla $f_{osc}=12MHz$).

PS2	PS1	PS0	pojemność licznika	czas zliczania (ms)
0	0	0	$2^{14} - 1$	16
0	0	1	$2^{15} - 1$	32
0	1	0	$2^{16} - 1$	64
0	1	1	$2^{17} - 1$	128
1	0	0	$2^{18} - 1$	256
1	0	1	$2^{19} - 1$	512
1	1	0	$2^{20} - 1$	1024
1	1	1	$2^{21} - 1$	2048

Podany wyżej opis działania bitów jest dość enigmatyczny i w kilku przypadkach wymaga dodatkowego komentarza. Działanie bitów PS0..PS2 jest oczywiste - ich stan określa pojemność licznika WDT. Stan bitu DISRTO pozwala na zadeklarowanie sposobu używania końcówki RST (kasowania mikrokontrolera). Gdy DISRTO = 1, to końcówka RST jest końcówką wejściową. Gdy DISRTO = 0, to w momencie kasowania układu przez WDT, końcówka RST jest ustawiana w stan jedynki logicznej - staje się końcówką typu wyjściowego. Dzięki takiemu rozwiązaniu, układ WDT może skasować nie tylko mikrokontroler ale i urządzenia peryferyjne systemu mikroprocesorowego jeżeli korzystają z sygnału RST.

Po wprowadzeniu mikrokontrolera w stan jałowy, praca licznika jest kontynuowana gdy bit WDIDLE = 0 i zatrzymywana, gdy WDIDLE = 1. W celu ochrony mikrokontrolera przed automatycznym skasowaniem, co pewien czas powinno następować wybudzenie mikrokontrolera ze stanu jałowego (np. przerwaniem), skasowanie licznika WDT i ponowne wprowadzenie mikrokontrolera w stan pracy jałowej.

Bit HWDT pozwala zadeklarować sposób pracy układu WDT - układ może pracować w tzw. trybie programowym gdy HWDT = 0 oraz w trybie sprzętowym, gdy HWPT = 1:

- w trybie sterowania programowego, układ licznika WDT jest uruchamiany gdy bit WDTEN = 0. Do kasowania licznika wystarczy, w takim przypadku, ustawienie bitu WSWRST (WSWRST = 1).

Po skasowaniu licznika, bit WSWRST jest kasowany automatycznie przez mikrokontroler w następnym cyklu maszynowym. W celu zatrzymania pracy WDT trzeba skasować bit WDTEN (WDTEN = 0). Zmiana stanu bitu WDTEN z 1 na 0 nie kasuje bieżącego stanu licznika.

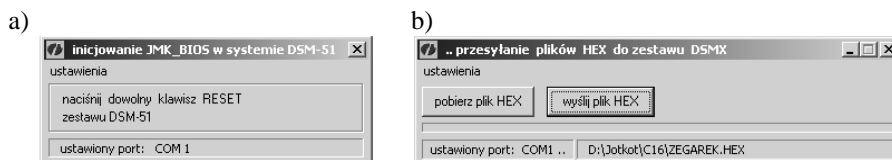
- w trybie sterowania sprzętowego, uruchomiony układ WDT można wyłączyć wyłącznie sygnałem zewnętrznego kasowania mikrokontrolera. Uruchomienie WDT można osiągnąć przez wpisanie do rejestru WDTRST (0A6h) dwu bajtów o wartościach 1Eh i 0E1h. Zapis bajtów powinien być wykonany dwoma, następującymi po sobie instrukcjami MOV. Po uruchomieniu WDT, bit WDTEN jest sprzętowo ustawiany w stan jedynki logicznej i nie można zmienić tego stanu - bit staje się bitem wyłącznie do odczytu. Bit WSWRST w trybie sprzętowym jest nieaktywny. Kasowanie stanu licznika wykonuje się przez kolejne wpisanie do rejestru WDTRST dwu bajtów o wartościach 1Eh i 0E1h.

1.3. Pracownia systemów mikroprocesorowych.

Do wykonania zadań, w laboratorium można wykorzystywać system mikroprocesorowy DSM-51 lub FTSM_51. Oba systemy wykorzystują mikrokontrolery z rodziny MCS-51: w systemie DSM-51 zastosowano odpowiednik mikrokontrolera 80C51 a w systemie FTSM_51 mikrokontroler AT89S8253 (odpowiednik 80C52). Zewnętrznie, oba systemy różnią się tym, że system DSM-51 jest wyposażony w szereg, zainstalowanych "na stałe" urządzeń I/O a system FTSM_51 ich nie posiada - posługuje się on urządzeniami wirtualnymi.

1.3.1. System DSM-51¹.

Budowa i działanie systemu DSM-51 oraz dokładne opisy sposobu programowania poszczególnych urządzeń I/O opisano, w wyczerpujący sposób, w podręczniku dołączonym do systemu [3]. W celu przystosowania systemu do innych zadań oraz uniknięcia bezpośredniego kopiowania treści przykładów podanych w wyżej wymienionym podręczniku, system DSM-51 poddano nieznacznej przeróbce sprzętowej a do pamięci ROM dopisano dodatkowe oprogramowanie, które dalej będzie określane umowną nazwą JMK_BIOS. W wyniku tych zmian, po włączeniu zasilania, system pracuje w sposób standardowy - jest oryginalnym systemem DSM-51. Po uruchomieniu programu narzędziowego SET_BIOS (set_bios.exe) i naciśnięciu dowolnego klawisza RESET, system przechodzi do nowego trybu pracy i od tego momentu jest obsługiwany przez JMK_BIOS. Wygląd programu SET_BIOS pokazano na rys.1.3.1a. Przyjęto, że system w trybie obsługiwanym przez JMK_BIOS zmienia nazwę na DSMX. W trybie DSMX zablokowany jest dostęp do oprogramowania oryginalnego DSM-51. Przejście w drugą stronę, do trybu DSM-51, jest możliwe wyłącznie przez wyłączenie i ponowne włączenie zasilania systemu.



Rys. 1.3.1. Wygląd pola pulpitu programów narzędziowych SET_BIOS (a) oraz HEX_SENDER (b).

Po wprowadzeniu systemu DSM-51 w tryb DSMX, łącze RS232 jest aktywne i pozwala na asynchroniczną wymianę informacji z parametrami transmisji: 19200 b/s, 8 bitów danej, 1 bit stopu - kontrola parzystości lub nieparzystości musi być wyłączona. System DSMX odpowiada na każdy wysłany z zewnątrz bajt. Jeżeli wysyłane bajty należą do zestawu tzw. komend, to system odpowiada bajtem o takiej samej wartości. Po wprowadzeniu wszystkich bajtów komendy (2 bajty) system odpowiada stosownym komunikatem.

¹ produkt f-my: "MicroMade"; ul. Wieniawskiego 16, 64-920 Piła;

Zestaw komend akceptowanych przez system DSMX podano w tabeli 1.3.1. W fazie 1 system odpowiada treścią komendy; w fazie 2 wysyłany jest komunikat kończący wykonanie komendy. W przypadku komendy X3, po fazie 2 system DSMX oczekuje na przesłanie mu zawartości pliku z rozszerzeniem HEX - treści programu, który powinien być uruchomiony w systemie. Po każdym bezbłędnym otrzymaniu rekordu pliku HEX, system wysyła znak kropki (faza 3). Po bezbłędnym otrzymaniu treści programu, system wysyła komunikat opisany fazą 4 i przekazuje sterowanie do wprowadzonego programu. Funkcje komendy X3 są wykorzystywane w programie narzędziowym HEX_SENDER (hex_sender.exe), który napisano w celu umożliwienia prostego przesyłania treści pliku HEX do systemu DSMX. Wygląd programu pokazano na rys.1.3.1b. Przed wysłaniem pliku należy nacisnąć przycisk RESET_ROM systemu DSMX.

Tabela 1.3.1. Wykaz komend akceptowanych przez DSMX.

komenda	faza	komunikat tekstowy odpowiedzi:
X0	1	X0
	2	Zestaw dydaktyczny DSMX; wersja BIOS dla II UMCS
X1	1	X1
	2	test obecności DSMX!
X2	1	X2
	2	oprogramowanie BIOS: JMK_BIOS, v.1.0.1
X3	1	X3
	2	oczekiwanie na kod HEX INTEL'a ..
	3 (kropka po każdym poprawnie odebranym rekordzie pliku HEX)
	4	sterowanie przekazano do RAM..

Tabela 1.3.2. Wykaz komunikatów na wyświetlaczu LCD.

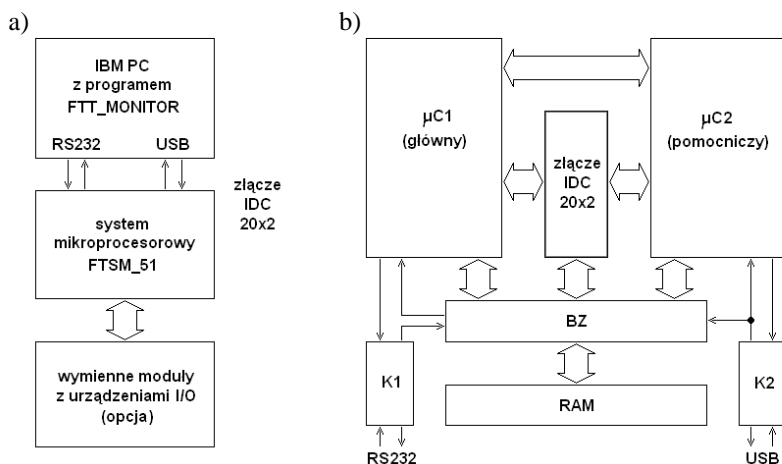
nr	komunikat tekstowy:
1	sterowanie DSMX z pamięci ROM
2	ładowanie pliku INTEL_HEX
3	sterowanie DSMX z pamięci RAM

W każdym przypadku, gdy wprowadzany do systemu DSMX bajt nie należy do treści komendy lub zaobserwowano błąd przesyłania pliku HEX, odpowiedzią DSMX jest sekwencja bajtów o wartościach: 35,10,13 i 46. Bajty te, wprowadzone w pole tekstowe generują tekst: "#." z kursorem przeniesionym do nowej linii - jest to ogólne oznaczenie wystąpienia błędu. Po wprowadzeniu systemu DSM-51 w tryb DSMX, po każdorazowym naciśnięciu przycisku RESET_ROM, przez łącze RS232 wysyłany jest komunikat tekstowy - system DSMX

"przedstawia się". Wysyłany komunikat jest identyczny do tego, wysyłanego po odebraniu komendy $X0$. Oprócz wysyłania komunikatów łączem RS232, bieżący status DSMX jest przekazywany do wyświetlacza LCD. Zestaw komunikatów wyświetlacza LCD pokazano w tabeli 1.3.2.

1.3.2. System FTSM_51¹.

System FTSM_51 jest mikroprocesorowym systemem dydaktycznym, w którym przyjęto inną, od powszechnie stosowanej, koncepcję pracy układu. System jest pozbawiony typowych dla systemów mikroprocesorowych urządzeń I/O, takich jak klawiatura, wskaźniki LED, wyświetlacz LCD itp. Urządzenia te, jako obiekty wirtualne, umieszczono na ekranie komputera IBM_PC - na pulpicie programu FTT_MONITOR (ftt_monitor.exe). Jednocześnie, dzięki zainstalowanemu w systemie FTSM_51 złączu IDC40, możliwe jest dołączanie do systemu rzeczywistych urządzeń I/O. Opis wyprowadzeń złącza IDC40 umieszczono w dodatku (Tabela 1.3.3). Po dołączeniu do systemu urządzenia rzeczywistego, w programie monitora dezaktywowany jest wirtualny odpowiednik tego urządzenia i system rozpoczyna bezpośrednią współpracę z urządzeniem. Jednocześnie, pomimo formalnej dezaktywacji obiektu wirtualnego, system "śledzi" wymianę informacji z tym obiektem a wynik obserwacji wprowadzany jest we właściwe miejsca obiektu wirtualnego. Pozwala to na bieżące kontrolowanie stanu urządzenia rzeczywistego.

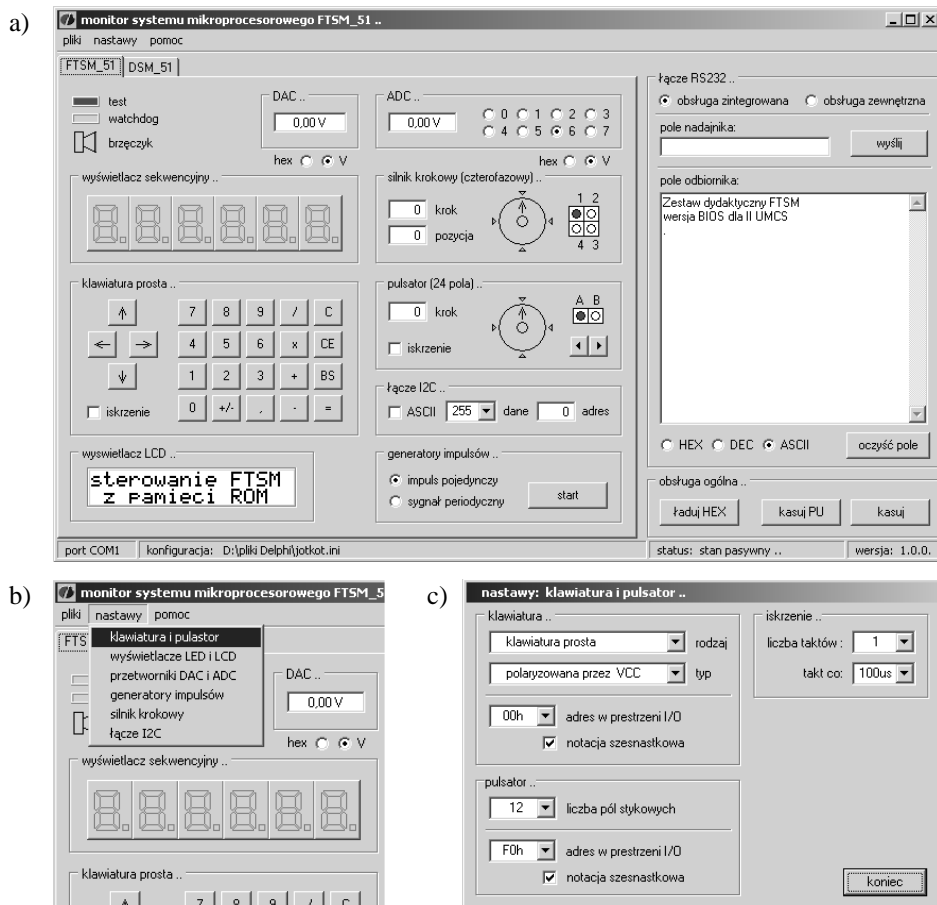


Rys. 1.3.2. Schemat zestawu laboratoryjnego z systemem FTSM_51 (a) oraz schemat blokowy struktury FTSM_51 (b).

Na rys.1.3.2b, w dużym uproszczeniu, pokazano strukturę systemu FTSM_51. System posiada dwa mikrokontrolery: $\mu C1$ i $\mu C2$, dwa łącza transmisji sze-

¹ produkt f-my: "FORTEST, Prywatna Pracownia Badawczo-Rozwojowa", ul. Bursztynowa 29/33a, 20-576 Lublin;

regowej: USB i RS232, pamięć statyczną RAM oraz złącze IDC, umożliwiające dołączenie do systemu zewnętrznych urządzeń I/O. Sygnały standardu RS232 i USB są wprowadzane i wyprowadzane z systemu za pośrednictwem konwerterów K1 i K2. Dostęp do złącza IDC, w sposób bezpośredni i pośredni, mają oba mikrokontrolery. Dostęp bezpośredni jest powiązany z przyłączeniem do złącza IDC wybranych końcówek portów I/O mikrokontrolera $\mu C1$ (port P1 i częściowo P3 - patrz opis złącza na stronie 144). Wszystkie, połączone ze złączem końcówki mikrokontrolera $\mu C1$ są również sprzężone z mikrokontrolerem $\mu C2$ co umożliwia śledzenie ich stanu. Dostęp pośredni, traktowany jako dostęp do urządzeń zewnętrznych I/O, jest realizowany za pośrednictwem bloku zarządzania (BZ). Blok ten umożliwia również dostęp obu mikrokontrolerów do pamięci RAM która służy do przechowywania programu użytkownika dla mikrokontrolera $\mu C1$ oraz symuluje obecność urządzeń I/O.



Rys. 1.3.3. Wygląd pulpitu programu FTT_MONITOR: karta systemu FTSM (a), wybieranie urządzenia w polu menu (b) oraz pole nastaw klawiatury i pulsatora (c).

System FTSM_51 musi pracować w bezpośrednim sprzężeniu z zewnętrznym komputerem, np. z IBM_PC. System powinien być połączony z IBM przez łącze USB i RS232 tak, jak to pokazano na rys. 1.3.2a. W systemie FTSM_51, linie RxD i TxD kanału RS232 są połączone do końcówek portu szeregowego mikrokontrolera głównego.μC1. Łącze USB jest wykorzystywane do zasilania systemu FTSM_51 oraz do przesyłania informacji sterujących systemem.

Opisana wyżej koncepcja działania systemu FTSM_51 pozwala na zdefiniowanie bardzo różnorodnego środowiska sprzętowego systemu. Środowisko to jest limitowane wyłącznie zawartością biblioteki urządzeń wprowadzonych do programu FTT_MONITOR. Dzięki temu, w systemie FTSM_51 możliwe jest programowanie obiektów w pełni rzeczywistych, np. systemu DSM-51. W takim przypadku, program napisany pod kontrolą systemu FTSM_51 ale dedykowany środowisku DSM-51, będzie działał poprawnie w systemie DSM-51. I na odwrót - program napisany dla systemu DSM-51 i przeniesiony do FTSM_51 będzie działał w tym systemie w sposób identyczny jak w DSM-51.

Na rys. 1.3.3 pokazano wygląd pulpitu programu FTT_MONITOR w kilku przykładowych konfiguracjach. Pomimo tego, że interfejs programu należy do grupy interfejsów intuicyjnych, w polu *pomoc* menu programu umieszczono krótkie opisy sposobu używania programu (dokładne informacje podano w instrukcji obsługi dołączonej do systemu FTSM_51).

Praca z systemem FTSM_51 sprowadza się do napisania programu i skompilowania go za pomocą, np. asemblera ASEM_51 - kod wynikowy kompilacji, w formacie HEX Intel'a, musi być przesłany do systemu za pomocą programu FTT_MONITOR. Przed rozpoczęciem pisania programu, opcją menu *nastawy* trzeba zadeklarować stan urządzeń, które będą używane - nastawy te powinno się zapisać opcją menu *pliki/zapisz*. Dzięki temu, następnym razem, wystarczy je tylko odczytać (*pliki/czytaj*) by wszystkie zadeklarowane nastawy były przywrócone. Pozostałe czynności obsługi systemu można ująć w kilku punktach:

- przesłanie programu użytkownika do FTSM wykonywane jest po naciśnięciu przycisku *ładuj HEX* - przesyłanie programu może być wykonane w dowolnym momencie pracy systemu; po odebraniu treści programu, bezwzględnie rozpoczyna się jego wykonywanie;
- naciśnięcie przycisku *kasuj PU* powoduje, że system rozpoczyna wykonywanie programu użytkownika od początku - w przypadku braku takiego programu, naciśnięcie przycisku niczego nie zmienia;
- jeżeli w ramach programu jest obsługiwane łącze RS232, to wybierając opcję *obsługa zintegrowana* można śledzić stan transmisji w polu *łącze RS232* - wybór opcji *obsługa zewnętrzna* powoduje, że transmisja może być obsługiwana przez inny program;
- naciśnięcie przycisku *kasuj* powoduje, że system przerywa wykonywanie programu użytkownika (jeżeli taki był wykonywany) i przechodzi do tzw. trybu pasywnego, wysyłając łączem RS232 swoją wizytówkę -

- w tym trybie działanie systemu sprowadza się do opisanego wyżej systemu DSM-51 z programem JMK_BIOS (DSMX);
- po uruchomieniu programu użytkownika, jego testowanie jest automatyczne
 - o poprawności działania programu świadczy zachowanie się wirtualnych lub rzeczywistych urządzeń I/O, które powinno być zgodne z oczekiwaniami;

1.3.3. Ćwiczenia laboratoryjne.

Programowanie komputerów można wykonać na wiele sposobów - za każdym razem można wczytywać się, w często skomplikowane opisy urządzeń I/O, co zajmuje bardzo dużo czasu. Wygodną metodą jest opracowanie szeregu procedur, które pozwalają zapomnieć o sprzęcie - ich automatyczne użycie zamienia sprzęt w zestaw bajtów umieszczonych w pamięci RAM.

W opisach podanych niżej zadań laboratoryjnych pojawiają się określenia zdefiniowane w rozdziale 1.2, który poświęcony był problemom programowania mikrokomputerów - zalecane jest zapoznanie się z przedstawionymi tam tezami.

Ze względu na niewielki rozmiar pamięci wewnętrznej RAM, w każdym przypadku, przy ocenianiu pracy studenta premiowane będzie oszczędne używanie tej pamięci.

Zadanie 1: generator zdarzeń z licznikiem T0.

Pojęcie *generator zdarzeń czasowych* lub prościej, generator zdarzeń, jest związane z problemem aktywacji określonych zdarzeń środowiska mikrokomputera, które powinny wystąpić w ściśle określonych momentach czasowych. Przykładami takich zdarzeń może być, np. obsługa wyświetlacza multiplexowanego LED (zadanie 3), obsługa klawiatury (zadanie 2) czy też potrzeba zmiany stanu zegara czasu rzeczywistego z częstotliwością 1Hz (zadanie 9).

Opis zadania.

Podprogram, który należy utworzyć w ramach niniejszego zadania powinien zapewnić obsługę licznika T0 z wykorzystaniem przerwań. Zaleca się, by podprogram nosił nazwę T0_OBSLUGA i był wykonywany co 1ms. Licznik T0 powinien pracować w trybie 1 jako czasomierz. Podprogram powinien wygenerować stany aktywne bitów flagowych T03_FLG, T020_FLG oraz T01000_FLG w odstępach czasu co 3, 20 i 1000ms. Poprawność uzyskania odcinków czasu 20ms i 1000ms, można sprawdzić poprzez przełączanie stanu zasilania diody wskaźnikowej LED, D1.

Uwagi o urządzeniach I/O i wykonywaniu zadania.

Problem aktywacji zdarzeń w ściśle określonych momentach czasowych nie występuje w przypadku gdy w systemie mikroprocesorowym dostępnych jest wiele układów czasowo-licznikowych i każdemu zdarzeniu można przypisać oddzielny układ. W mikrokontrolerze 80C51, do dyspozycji są dwa układy

czasowo-licznikowe: T0 i T1. Ponieważ licznik T1 jest wykorzystywany do takowania portu transmisji szeregowej, do dyspozycji pozostaje wyłącznie licznik T0. Licznik T0 może poprawnie wygenerować tylko 1 zdarzenie typu czasowego. Jeżeli licznik T0 zostanie przypisany do najczęściej pojawiającego się zdarzenia periodycznego, to czas przez ten licznik wyznaczany będzie najkrótszy - można w takim przypadku mówić o czasie bazowym - czasie systemowym mikrokomputera. Pojęcia tego nie należy mylić z czasem wyznaczanym przez cykl zegarowy lub cykl maszynowy systemu mikroprocesorowego. Zakładając dalej, że pozostałe odcinki czasu będą wielokrotnością czasu systemowego, można je utworzyć w sposób programowy - w ramach obsługi zdarzenia wywołanego przez licznik T0. Wyznaczanie innego odcinka czasu, w takim przypadku, jest związane wyłącznie z obliczaniem liczby zdarzeń generowanych przez licznik T0. Po przekroczeniu określonej liczby zliczeń, można przekazać informację w wystąpieniu innego zdarzenia czasowego przez ustawienie flagi tego zdarzenia w stan aktywny.

Problemy do rozstrzygnięcia:

- poprawne rozdzielanie zadań obsługi licznika T0 pomiędzy przerwanie a program uzupełniający obsługę zdarzenia.
- **Słowa kluczowe:**
 - T0_OBSLUGA - nazwa podprogramu;
 - T03_FLG - flaga zdarzenia generowanego co 3ms;
 - T020_FLG - flaga zdarzenia generowanego co 20ms;
 - T01000_FLG - flaga zdarzenia generowanego co 1s;

Uwagi dodatkowe:

- częstotliwość własna generatora cyklu zegarowego wynosi: 11,0592 MHz;
- nic nie stoi na przeszkodzie by zdefiniować inne odcinki czasu - dobór czasu, oznaczenie flag i ich lokalizacja w pamięci RAM pozostawiono decyzji studenta.

Zadanie 2: obsługa klawiatury prostej i multipleksowanej.

Klawiatura systemu mikroprocesorowego umożliwia wprowadzenie informacji do systemu za pośrednictwem zespołu przycisków mechanicznych. Ze względu na sposób pozyskiwania informacji o stanie przycisków, klawiaturę można podzielić na tzw. klawiaturę prostą (dostęp bezpośredni) i multipleksowaną (o dostępie sekwencyjnym). Z punktu widzenia mikrokontrolera, każdy przycisk jest elementem linii wejściowej, której stan powinien być odczytany.

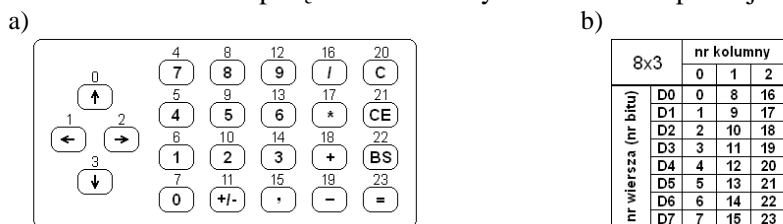
Opis zadania.

W ramach ćwiczenia trzeba napisać podprogram obsługi klawiatury - zaleca się, by podprogram nosił nazwę KBD_OBSLUGA. Podprogram powinien być wywoływany co pewien, ściśle określony czas z obszaru pętli programowej.

Podprogram powinien rozpoznać fakt zmiany stanu klawiatury i ustawić flagę tego zdarzenia - KBD_FLG. W przypadku zaobserwowania naciśnięcia przycisku klawiatury, do rejestru o nazwie KBD_KOD powinien być wprowadzony kod naciśniętego przycisku. Kod przycisku klawiatury powinien być numerem przycisku, zapisanym w naturalnym kodzie binarnym. W przypadku zwolnienia nacisku na przycisk, do rejestru KBD_KOD powinna być wpisana wartość 255 (0FFh). Podprogram obsługi klawiatury powinien zapewnić eliminację problemu odbijania się styków przycisków klawiatury i zmieniać stan flagi KBD_FLG oraz rejestru KBD_KOD jedynie w przypadku jednoznacznego pozyskania informacji o stanie klawiatury.

Uwagi o urządzeniach I/O i wykonywaniu zadania.

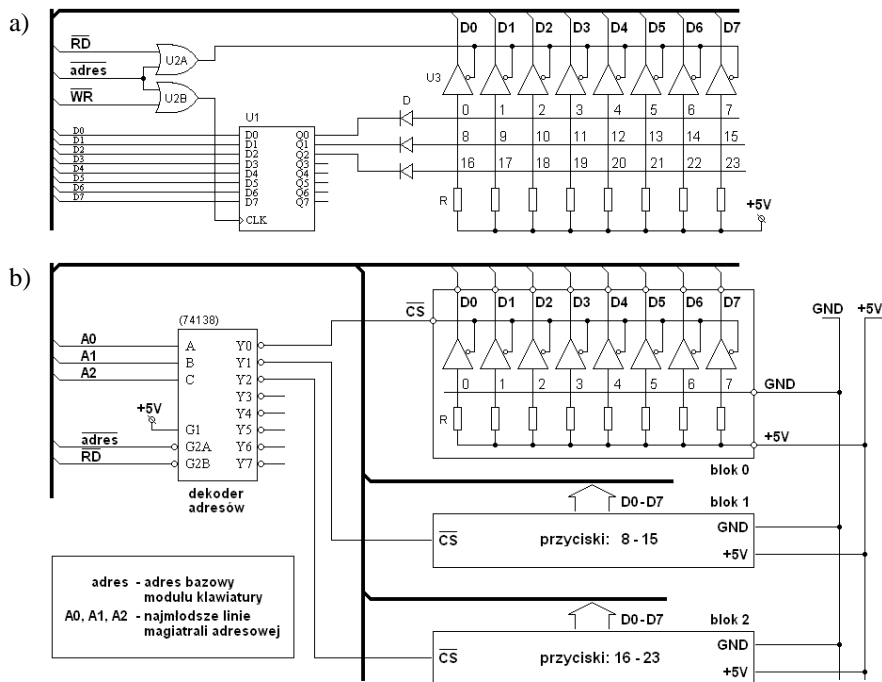
System mikroprocesorowy FTSM_51 jest wyposażony w 24 przyciskową, wirtualną klawiaturę uniwersalną. Uniwersalność klawiatury jest związana z możliwością jej rekonfiguracji za pośrednictwem okna nastaw w programie FTT_MONITOR. Klawiatura może być klawiaturą prostą lub multipleksowaną z różną konfiguracją połączeń wewnętrznych. Wygląd zewnętrzny klawiatury pokazano na rys.1.3.4a - na rysunku zaznaczono numery poszczególnych przycisków. Na rys.1.3.4b pokazano mapę adresową wszystkich przycisków, zorganizowanych w trybie "8x3". Pierwsza cyfra określa liczbę przycisków, których stan można określić przez jednorazowy odczyt danych z pola adresowego klawiatury. Druga cyfra określa liczbę odczytów stanu klawiatury, która musi być wykonana w celu poznania stanu wszystkich przycisków klawiatury. W przypadku dołączenia do systemu FTSM_51 modułu rozszerzenia, dane w oknie nastaw mogą ulec modyfikacji i będą odzwierciedlały architekturę dołączonego urządzenia. Na rysunku 1.3.5 pokazano schematy połączeń wewnętrznych klawiatury, ustawionej w tryb "8x3". Na rysunku 1.3.5a przedstawiono schemat połączeń stosowany w klawiaturze multipleksowanej a na rysunku 1.3.5b - schemat połączeń stosowany w klawiaturze prostej.



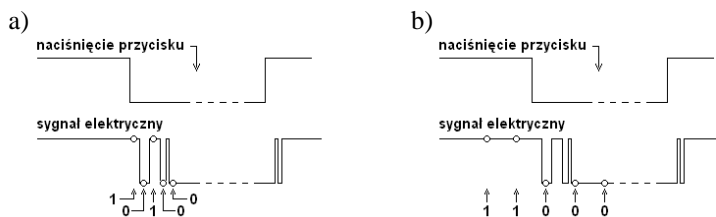
Rys. 1.3.4. Wygląd klawiatury uniwersalnej FTSM_51 (a) oraz mapa adresowa przycisków klawiatury dla trybu "8x3"(b).

W przypadku klawiatury multipleksowanej, odczyt stanu klawiatury następuje spod adresu bazowego, który jest ustawiany w oknie nastaw programu FTT_MONITOR. Pod ten sam adres powinien być wpisywany bajt testujący stan klawiatury.

W przypadku klawiatury prostej, pierwszy odczyt stanu klawiatury następuje spod adresu bazowego, który jest ustawiany w oknie nastaw programu FTT_MONITOR. Następne odczyty są wykonywane spod kolejnych adresów, o coraz to większej wartości (z krokiem co 1). Liczba adresów odczytu jest determinowana organizacją połączeń wewnętrznych klawiatury. Dla trybu "8x3" odczyt następuje spod 3 kolejnych adresów.



Rys. 1.3.5. Schematy klawiatur o organizacji 8x3: klawiatura multipleksowana (a); klawiatura prosta (b).



Rys. 1.3.6. Problem odbijania się styków: próbkowanie częste (a); próbkowanie rzadkie (b).

W przypadku posługiwania się klawiaturą, której przyciskami są elementy mechaniczne może wystąpić zjawisko odbijania się styków przycisku od siebie. Zjawisko jest związane z własnościami dynamicznymi ruchomych elementów przycisku. Jest ono szczególnie dobrze widoczne w momencie zwierania styków. Opisywaną sytuację przedstawiono na rys.1.3.6. Po naciśnięciu przycisku

następuje krótkotrwałe zwieranie i rozwieranie styków przycisku przez okres ok. kilkuset μ s (rys.1.3.6a). Zbyt częste testowanie stanu przycisku powoduje, że zjawisko to może być zaobserwowane i fałszywie zinterpretowane jako wielokrotne naciśnięcie przycisku. W przypadku testowania rozciągniętego w czasie (rys. 1.3.6b) problem rejestracji drgań styków praktycznie nie istnieje. Dobrym rozwiązaniem jest podejmowanie decyzji o stanie przycisku po wielokrotnym testowaniu jego stanu, wykonywanym ze stosunkowo dużym odstępem czasu. Zaobserwowanie co najmniej dwukrotnego, kolejnego powtórzenia się tego samego stanu, w dwu kolejnych cyklach testowania, upoważnia do przypisania przyciskowi tego stanu.

Problemy do rozstrzygnięcia:

- określenie optymalnego czasu wywoływania podprogr. KBD_OBSLUGA;
- likwidacja problemu odbijania się styków;
- problem jednoczesnego naciśnięcia wielu przycisków.

Słowa kluczowe:

KBD_OBSLUGA	- nazwa podprogramu obsługi klawiatury;
KBD_KOD	- bufor na kod naciśniętego przycisku;
KBD_FLG	- flaga sygnalizująca nowy stan klawiatury.

Uwagi dodatkowe:

- do wykonania zadania należy się posłużyć modułem T0_OBSLUGA (patrz zadanie 1), którego flaga T03_FLG lub T020_FLG (lub inna) wskaże właściwy moment wywołania podprogramu KBD_OBSLUGA;
- po wprowadzeniu do obszaru pętli programowej procedury automatycznej obsługi klawiatury, przekształca się ona z urządzenia I/O do rejestru pamięci RAM, KBD_KOD. Od tego momentu, każde inne zdarzenie, które chce pozyskać informację o stanie klawiatury, może to osiągnąć po zaobserwowaniu aktywnego stanu flagi KBD_FLG.

Zadanie 3: obsługa wyświetlacza multipleksowanego LED.

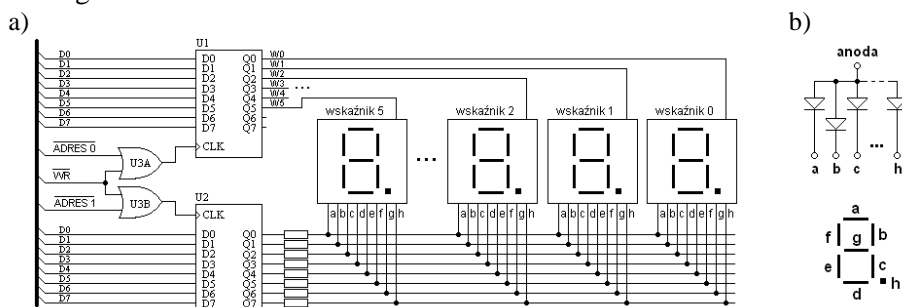
Wyświetlacz LED jest układem I/O, który pozwala na wyświetlanie informacji za pośrednictwem zestawu, tzw. 7-segmentowych, diodowych wskaników LED (ang. Light Emiting Diode). Wyświetlacze LED występują w dwu podstawowych odmianach jako wyświetlacze statyczne lub wyświetlacze dynamiczne. Te ostatnie są najczęściej nazywane wyświetlaczami multipleksowanymi lub sekwencyjnymi. Wyświetlacze dynamiczne, w odróżnieniu od wyświetlaczy statycznych, charakteryzują się bardzo prostą budową i zajmują w przestrzeni adresowej bardzo mało miejsca (2 adresy w przypadku wyświetlacza z 8 elementami LED). Ze względu na zastosowane elementy, wyświetlacze występują w dwu podstawowych odmianach: wyświetlacze z elementami LED ze wspólną katodą lub wspólną anodą.

Opis zadania.

W ramach niniejszego zadania trzeba napisać podprogram, który pozwoli na automatyczne wprowadzenie danych do modułu wyświetlacza LED. Zaleca się, by podprogram nosił nazwę LED_OBSLUGA.

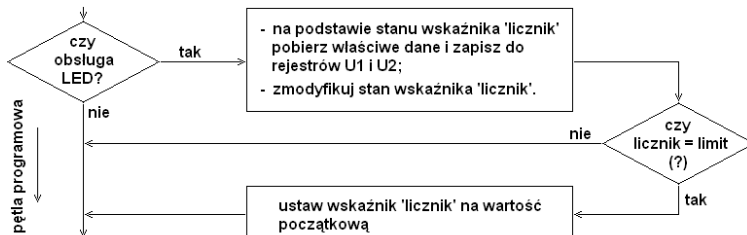
Uwagi o urządzeniach I/O i wykonywaniu zadania.

System mikroprocesorowy FTSM_51 umożliwia wybór typu wskaźnika multipleksowanego LED za pośrednictwem okna nastaw w programie FTT_MONITOR. Wskaźnik może być zbudowany z elementów LED ze wspólną anodą lub wspólną katodą. W oknie nastaw programu FTT_MONITOR można zadeklarować adres bazowy dla modułu - drugi adres jest o 1 większy od adresu bazowego.



Rys. 1.3.7. Uproszczony schemat blokowy 6-elementowego wyświetlacza dynamicznego LED (a) oraz schemat elektryczny elementu 7-segmentowego ze wspólną anodą i opisem jego topografii (b).

Na rysunku 1.3.7 pokazano typową konstrukcję wyświetlacza multipleksowanego (dynamicznego) opartego o 6 elementów LED. Praca modułu wyświetlacza polega na sukcesywnym wprowadzaniu danej do wyświetlania i określeniu wskaźnika, który ma tę daną wyświetlać - w danym momencie tylko jeden ze wskaźników wyświetla przekazaną mu informację. W przypadku odpowiednio szybkiej wymiany danych, użytkownik będzie postrzegał moduł wyświetlacza jako strukturę, która w sposób jednolity wyświetla wprowadzoną do niej informację.



Rys. 1.3.8. Typowy schemat obsługi wyświetlacza multipleksowanego LED.

Wyświetlacz multipleksowany LED powinien być obsługiwany w sposób automatyczny przez podprogram LED_OBSLUGA. Ponieważ obsługa wyświetlacza LED sprowadza się do przepisывania bloku danych do modułu LED,

podprogram LED_OBSLUGA jest procedurą kopiowania danych. Zaleca się, by dane dla rejestru U1 były kopiowane z pola pamięci programu o nazwie LED_NUMER a dane dla rejestru U2, z pola wewnętrznej pamięci RAM o nazwie LED_DANE. Do wskazywania bajtów w obu polach pamięci należy użyć bajtu licznika o nazwie LED_LICZNIK. Schemat działania podprogramu może wyglądać tak, jak pokazano to na rys.1.3.8.

Problemy do rozstrzygnięcia.

- dobór właściwego czasu odświeżania stanu wyświetlacza multipleksowanego;

Słowa kluczowe:

LED_OBSLUGA	- nazwa podprogramu obsługi wskaźnika LED;
LED_DANE	- adres początku pola RAM na dane dla U2;
LED_NUMER	- adres początku pola ROM na dane dla U1;
LED_LICZNIK	- adres bajtu licznika;

Uwagi dodatkowe:

- do wykonania zadania należy wykorzystać podprogram T0_OBSLUGA z zadania 1;
- po wprowadzeniu do obszaru pętli programowej procedury automatycznej obsługi wyświetlacza LED, wyświetlacz przekształca się z urządzenia I/O w pole pamięci RAM. Od tego momentu, każde inne zdarzenie, które musi przekazać dane do wyświetlania na wyświetlaczu LED, wprowadza te dane do pamięci RAM - dane zostaną wyświetlone w sposób automatyczny.

Zadanie 4: obsługa wyświetlacza LCD.

Wyświetlacz LCD jest układem I/O, który pozwala na wyświetlanie informacji za pośrednictwem modułu ciekłokrystalicznego. Wyświetlacze LCD występują w dwu podstawowych odmianach: wyświetlacze graficzne i wyświetlacze tekstowe. Wyświetlacze graficzne pozwalają decydować o stanie każdego punktu ekranu; wyświetlacze tekstowe umożliwiają wyświetlenie tekstu zbudowanego z predefiniowanych wzorców - znaków kodu ASCII. Zadanie w laboratorium jest skierowane na obsługę wyświetlacza tekstowego LCD.

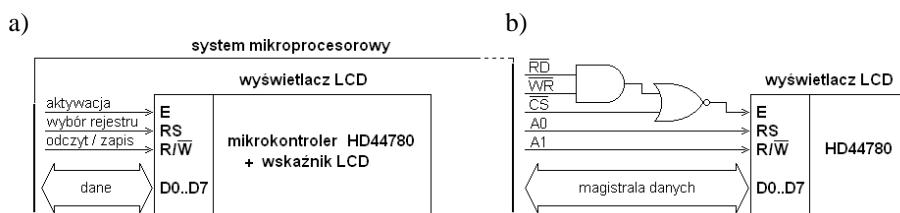
Opis zadania.

W ramach niniejszego zadania trzeba napisać podprogram, który pozwoli na wprowadzenie bloku danych do tekstowego modułu LCD. Blok danych może się znajdować w wewnętrznej pamięci RAM lub w pamięci programu. Zaleca się, by podprogram nosił nazwę LCD_OBSLUGA. Rozpoczęcie pracy podprogramu powinno być sygnalizowane przez właściwe ustawienie flagi LCD_FLG - flaga ta powinna być kasowana przez podprogram LCD_OBSLUGA w momencie, gdy wszystkie dane z bloku danych zostaną przesłane do modułu wyświetlacza.

Dodatkowo, w celu zainicjowania obsługi modułu LCD przez inne zdarzenia, w ramach omawianego zadania trzeba napisać podprogram pomocniczy o nazwie LCD_INI.

Uwagi o urządzeniu I/O.

Stosowane w systemach mikroprocesorowych FTSM_51 i DSM-51 moduły wyświetlaczy tekstowych LCD oparte są o specjalizowany mikrokontroler, HD44780. Kontroler HD44780, w swojej strukturze, posiada 3 rodzaje pamięci: pamięć ROM (CG_ROM - ang. Character Generator ROM), pamięć RAM (DD_RAM, ang. Display Data RAM) oraz dodatkową pamięć RAM (CG_RAM, ang. Character Generator RAM). Pamięć DD_RAM jest bezpośrednio sprzężona z polem tekstowym LCD - każdy rejestr pamięci przechowuje informację dla pojedynczego pola LCD. W rejestrach DD_RAM jest przechowywana informacja o wyglądzie (kroju) czcionki - informacja ta, przeniesiona na pole wskaźnika powoduje wyświetlenie wskazanej czcionki. Pamięć CG_ROM przechowuje informację o wzorcach kroju czcionki. To z tej pamięci pobierany jest wzorzec czcionki. W momencie zapisu danej do modułu LCD, kod danej wskazuje na wzorzec czcionki - wzorzec ten jest przepisywany do pamięci DD_RAM i jest wyświetlany. Czcionki, których kody są liczbami z zakresu 32..126 (32h..7Eh), pokrywają się z kodami ASCII. Ponieważ zestaw wzorców znaków pamięci CG_ROM nie obejmuje znaków narodowych (za wyjątkiem grupy znaków japońskich), kontroler umożliwia zdefiniowanie tych znaków i zapamiętanie w pamięci CG_RAM - liczba znaków jest ograniczona do 8 a kody tych znaków muszą zawierać się w zakresie 0..7 (lub 8..15). Ponieważ pamięć CG_RAM jest pamięcią ulotną, definiowanie znaków narodowych powinno się odbywać po każdym uruchomieniu systemu mikroprocesorowego.



Rys. 1.3.9. Linie interfejsowe modułu LCD (a) i sposób włączenia modułu do systemu mikroprocesorowego (b).

Kontroler HD44780 może obsłużyć 80 pól znakowych LCD, zorganizowanych w pojedynczą linię lub 2 linie tekstowe. W przypadku organizacji jednoliniowej, poszczególne pola znakowe są adresowane od 00h do 4Fh (0..79). W przypadku organizacji dwuliniowej, poszczególne pola znakowe są adresowane od 00h do 27h (0..39) w przypadku pierwszej linii oraz od 40h do 67h (64..103) dla drugiej linii. Po przekroczeniu adresu pola znakowego, w przypadku organizacji jednoliniowej, adres pola zmienia się automatycznie z 4Fh na 00h (lub odwrotnie). W przypadku organizacji dwuliniowej, adres pola zmienia się automatycznie z 27h na 40h (skok z końca pierwszej linii na początek drugiej)

linii) oraz z 67h na 00h (skok z końca drugiej linii na początek pierwszej linii). W przypadku gdy adres jest zmniejszany, automatyczna zmiana adresów wynosi, odpowiednio z 00h na 67h oraz z 40h na 27h.

Tabela 1.3.3. Zestaw komend sterujących kontrolera HD44780.

RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	opis oryginalny [14]	nr instr.	czas dostępu ¹
0	0	0	0	0	0	0	0	0	1	Clear display	1	
0	0	0	0	0	0	0	0	1	x	Return home	2	1,52ms
0	0	0	0	0	0	0	1	I/D	S	Entry mode set	3	37μs
0	0	0	0	0	0	1	D	C	B	Display on/off control	4	37μs
0	0	0	0	0	1	S/C	R/L	x	x	Cursor or display shift	5	37μs
0	0	0	0	1	DL	N	F	x	x	Function set	6	37μs
0	0	0	1	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	Set CGRAM address	7	37μs
0	0	1	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	Set DDRAM address	8	37μs
0	1	BF	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	Read Busy Flag and Address	9	0μs

Z punktu widzenia systemu mikroprocesorowego, kontroler HD44780 jest reprezentowany przez 2 rejestry. Pierwszy z rejestrów nosi nazwę rejestru sterującego i można do niego zapisać 8 różnych danych, nazywanych instrukcjami. Wykaz instrukcji pokazano w tabeli 1.3.3. Drugi rejestr modułu LCD jest rejestrem danych. Za pośrednictwem tego rejestru wprowadza się do modułu kody znaków, które będą wyświetlane w polu wskaźnika LCD. Kontroler HD44780 może być dołączony do systemu mikroprocesorowego np. tak, jak to pokazano na rys. 1.3.9. Jeżeli pod symbolem CS będzie się ukrywało pole adresowe przypisane wyświetlaczowi LCD i najmłodszym adresem będzie adres *lcd_adr*, to dostęp do poszczególnych funkcji kontrolera będzie możliwy przez wpisywanie lub odczytywanie danych pod adresy pokazane w tabeli 1.3.4.

Tabela 1.3.4. Adresowanie modułu LCD z kontrolerem HD44780.

adres	RS	R/W	opis
lcd_adr	0	0	zapisywanie instrukcji
lcd_adr + 1	0	1	odczyt bajtu statusowego
lcd_adr + 2	1	0	zapisywanie danej do DD_RAM lub CG_RAM
lcd_adr + 3	1	1	odczytywanie danej z DD_RAM lub CG_RAM

Opis działania instrukcji sterujących z tabeli 1.3.3.

1. kasowanie danych pola DD_RAM: kasowanie odbywa się poprzez wpisywanie do wszystkich rejestrów pamięci DD_RAM wzorca kodu spacji (20h).

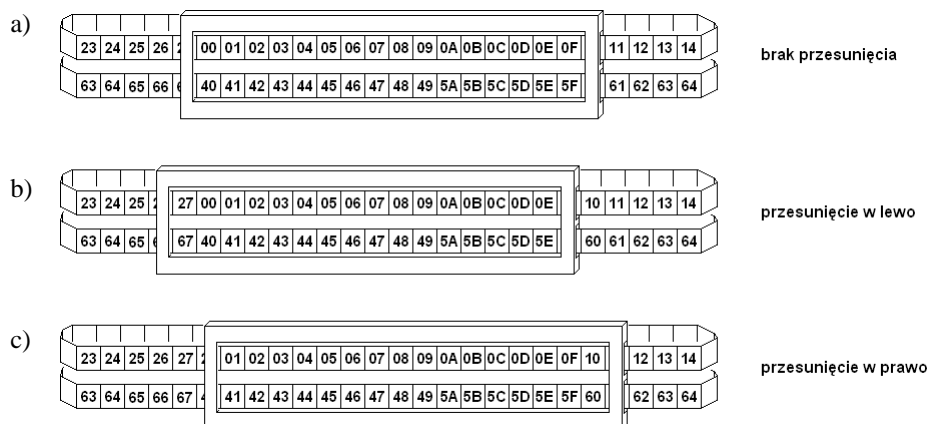
¹ czas maksymalny wykonywania instrukcji dla $f_{OSC} = 270 \text{ kHz}$ (HD44780U, [14]).

Rejestr adresowy pamięci DD_RAM (ang. address counter, AC) ustawiany jest na wartość 0 (pozycja kursora w lewym położeniu i górnej linii (w przypadku modułu z wieloma wierszami). Bit I/D jest ustawiany, a bit S nie jest zmieniany (patrz instrukcja nr 3);

2. to samo co (1) za wyjątkiem kasowania danych - dane w pamięci DD_RAM nie są kasowane;
3. bit **I/D** (ang. increment or decrement mode): gdy I/D=1 to adres pola DD_RAM jest zwiększany o 1 po każdorazowym zapisie lub odczycie danej; gdy I/D=0 to adres pola DD_RAM jest zmniejszany o 1 - identyczne działanie jest przypisane do pola CG_RAM;
bit **S** (ang. shift mode): gdy S=1 to pole wyświetlanych danych jest przesuwane w lewo w przypadku, gdy I/D=1 (obrót w lewo). Gdy I/D=0 to przesunięcie jest wykonywane w prawą stronę (obrót w prawo - patrz rys.1.3.10). Pole wyświetlania nie jest przesuwane gdy S=0. Gdy S=1 i wykonywany jest odczyt danej z DD_RAM to pole wyświetlania też nie jest przesuwane. W przypadku pamięci CG_RAM pole wyświetlania nie jest przesuwane ani przy zapisie, ani przy odczycie danej do tej pamięci;
4. bit **D** (ang. display mode): gdy D=1 to dane z pamięci DD_RAM są wyświetlane; gdy D=0 to wyświetlanie danych jest zablokowane;
bit **C** (ang. cursor mode): gdy C=1 to pod znakiem wskazywanym przez adres pola DD_RAM jest wyświetlany kreska kursora; gdy C=0 to kursor nie jest wyświetlany;
bit **B** (ang. cursor blinks mode): gdy B=1 to znak kursora jest znakiem mrugającym;
5. przesuwanie kursora lub pola wyświetlanych danych w przypadku, gdy trzeba dokonać takiego przesunięcia bez zapisu lub odczytu danych (patrz instrukcja nr 3). W przypadku trybu dwuliniowego, przesuwanie pola wyświetlania dotyczy obu linii jednocześnie. Przesuwanie kursora w trybie dwuliniowym spowoduje przesunięcie kursora do początku drugiej linii jeżeli przekroczony zostanie adres końca pierwszej linii. Zawartość licznika adresów pamięci DD_RAM (AC) nie ulega zmianie gdy instrukcja dotyczy przesuwania pola wyświetlanych danych;
bit **S/C**: gdy S/C=1 to przesuwanie dotyczy pola wyświetlanych danych; gdy S/C=0 to przesuwanie dotyczy kursora;
bit **R/L** (ang. right or left): gdy R/L=1 to kursor lub pole wyświetlanych danych jest przesuwane w prawo; gdy R/L=0 to wykonywane jest przesunięcie w lewą stronę;
6. bit **DL** (ang. data length): gdy DL=1 to wymiana informacji z systemem mikroprocesorowym jest wykonywana 8-bitową magistralą danych; gdy DL=0 to wymiana informacji jest wykonywana magistralą 4-bitową (tryb nieużywany w laboratorium);
bit **N** (ang. number): gdy N=1 to wyświetlacz pracuje w trybie dwuliniowym; gdy N=0 to wyświetlacz obsługuje 1 linię;

bit **F** (ang. font): gdy $F=0$ to znaki tekstowe mają rozdzielczość punktową 5×7 ; gdy $F=1$ to rozdzielczość punktowa wynosi 5×10 (tryb nieużywany w laboratorium);

7. ustawianie adresu w polu pamięci CG_RAM: po wykonaniu tej operacji możliwe jest wpisywanie lub odczytywanie danych z pamięci CG_RAM (pamięć znaków nietypowych, np. narodowych);
8. ustawianie adresu w polu pamięci DD_RAM;
9. odczytana z rejestru kontrolnego dana (bajt statusu) niesie informację o stanie modułu LCD: wykonywanie jakichkolwiek zapisów lub odczytów jest zabronione gdy bit **BF** jest jedynką logiczną (oczywiście nie dotyczy to odczytywania bajtu statusu). Gdy $BF=0$ to adres przekazywany przez pozostałe bity bajtu statusu jest bieżącym adresem pola DD_RAM (AC).



Rys. 1.3.10. Przesuwanie pola wyświetlanych danych o 1 pozycję: b) przesunięcie w lewo (b) oraz przesunięcie w prawo (c).

Po włączeniu zasilania, moduł wyświetlacza powinien być odpowiednio zainicjowany przez wpisanie do rejestru kontrolnego kilku instrukcji. Przed każdym wpisaniem instrukcji powinno się sprawdzić stan flagi BF. Zaleca się by proces uruchamiania przebiegał wg następującego algorytmu:

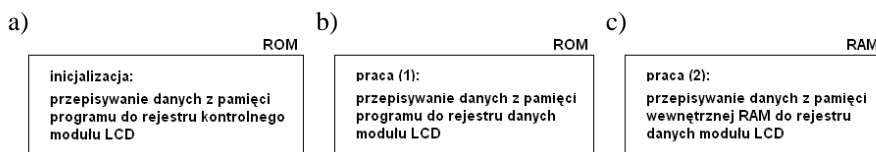
- instrukcja 1 - kasowanie stanu wyświetlacza;
- instrukcja 6 - ustawianie sposobu transmisji danych i sposobu wyświetlania: $DL=1$ (8-bitowa wymiana informacji), $N=1$ (wyświetlanie dwuliniowe), $F=0$ (rozdzielczość wyświetlania znaków: 5×7 punktów);
- instrukcja 4 - ustawianie sposobu wyświetlania (c.d.): $D=1$ (wyświetlanie włączone); C i B wg uznania;
- instrukcja 3 - ustawienie ruchu kursora i przesuwania bloku danych: I/D oraz S wg uznania ale dla typowej współpracy z modułem zalecane jest ustawienie $I/D=1$ oraz $S=0$.

Jak już wspomniano, sterownik HD44780 umożliwia zdefiniowanie 8 nietypowych wzorców znakowych, np. znaków narodowych. Znaki są przechowywane w pamięci CG_RAM. Dostęp do pamięci CG_RAM umożliwia instrukcja nr 7. Instrukcja ta wskazuje adres w polu CG_RAM, do którego można zapisać (lub odczytać) bajt danej. Ponieważ każdemu znakowi jest przyporządkowane 8 kolejnych bajtów pamięci, młodsza część adresu (bity $A_2..A_0$) określa numer bajtu we wzorcu znaku. Starsza część adresu (bity $A_5..A_3$) wskazuje numer wzorca znaku - ten numer powinien być podawany jako kod znaku narodowego w przypadku chęci przeniesienia znaku do pamięci DD_RAM (do pola wyświetlacza). Sposób budowania znaków pokazano w dodatku, w tabeli 1.3.2. Każdy bit o wartości logicznej 1 będzie powodował zaczernienie punktu w polu LCD.

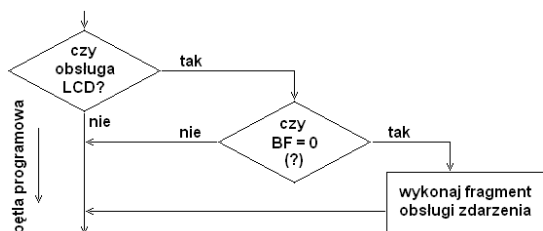
Uwagi o wykonywaniu zadania.

System mikroprocesorowy FTSM_51 umożliwia wybór typu wirtualnego wskaźnika (modułu) LCD za pośrednictwem okna nastaw w programie FTT_MONITOR. Wszystkie umieszczone w bazie programu moduły LCD są sterowane kontrolerem HD44780. W oknie nastaw programu FTT_MONITOR można również zadeklarować adres bazowy dla modułu - pozostałe adresy modułu zdefiniowane są w tabeli 1.3.4. W przypadku dołączenia do systemu FTSM_51 modułu rozszerzenia, dane w oknie nastaw mogą ulec modyfikacji i będą odzwierciedlały architekturę dołączonego urządzenia.

Wyświetlacz tekstowy LCD powinien być obsługiwany przez podprogram LCD_OBSLUGA. Podprogram ten powinien umożliwić zarówno proces inicjalizacji pracy modułu LCD jak i jego standardową pracę. Można zauważyć (patrz opis kontrolera modułu LCD), że praktycznie w każdym przypadku, poza procesem testowania stanu modułu, jest on odbiornikiem danych. Ponieważ obsługa wyświetlacza LCD sprowadza się do przepisywania bloku danych do modułu, podprogram LCD_OBSLUGA jest procedurą kopiowania danych. Kopiowanie to jest dodatkowo uproszczone przez fakt, że docelowe miejsce kopiowania jest zawsze takie samo - jest nim rejestr kontrolny lub rejestr danych modułu LCD. Ze względu na umiejscowienie źródła danych do kopiowania i miejsca docelowego kopiowania, obsługa modułu LCD może być rozbita na 2 elementy (2 tryby), które schematycznie pokazano na rysunku 1.3.11 - kopiowanie z pamięci programu (rysunek 1.3.11a i b) oraz kopiowanie z pamięci wewnętrznej RAM (rysunek 1.3.11c). Warto, w takim przypadku, wskazać programowi LCD_OBSLUGA źródło danych, np. za pośrednictwem bitu LCD_TRYB. Kopiowanie z pamięci ROM jest najczęściej stosowane do wprowadzenia nowego komunikatu w pełne pole wyświetlacza LCD. Kopiowanie z pamięci RAM stosuje się zwykle w celu wymiany informacji we wskazanym fragmencie pola LCD.



Rys. 1.3.11. Trzy tryby obsługi wyświetlacza LCD: inicjalizacja modułu (a), przepisywanie danych z ROM (b) oraz przepisywanie danych z RAM (c).



Rys. 1.3.12. Schemat typowej obsługi modułu LCD.

Ponieważ przed jakimkolwiek zapisem danej do modułu LCD wymagane jest przetestowanie jego stanu, typowy schemat działania może wyglądać tak, jak pokazano to na rys.1.3.12 - schemat ten jest identyczny dla każdego trybu obsługi. Zaleca się, by w trakcie pojedynczego cyklu obsługi pętli programowej, do modułu LCD przesyłać pojedynczy bajt z bloku danych. Dodatkowo zaleca się, by flaga zdarzenia (LCD_FLG) była bajtem. W takim przypadku można założyć, że gdy LCD_FLG = 0 to nie ma potrzeby obsługi modułu LCD - gdy LCD_FLG > 0 to taka obsługa jest konieczna a stan bajtu LCD_FLG wskazuje tryb pracy podprogramu obsługi.

Podprogram LCD_INI powinien umożliwić rozpoczęcie obsługi wyświetlacza LCD. Podprogram LCD_OBSLUGA powinien otrzymać informację o trybie pracy, o adresie źródła danych i liczbie kopiowanych danych. Zaleca się, by miejsce danych w pamięci RAM było stałe (wydzielone pole RAM) i początkowy adres tego miejsca był określony etykietą LCD_DANE. Informacja o liczbie bajtów do kopiowania powinna być przekazana przez rejestr ACC; przez rejestr B powinno się wprowadzić informację o numerze trybu pracy, a przez DPTR - informację o adresie źródła danych do kopiowania. Dobór rejestrów pomocniczych dla podprogramu LCD_OBSLUGA i ich lokalizację w pamięci RAM pozostawiono decyzji studenta.

Problemy do rozstrzygnięcia.

- dobór rejestrów pomocniczych dla podprogramu LCD_OBSLUGA i ich lokalizacja w pamięci RAM;

Słowa kluczowe:

LCD_OBSLUGA - nazwa podprogramu obsługi modułu LCD;
 LCD_INI - nazwa podprogramu inicjującego obsługę LCD;

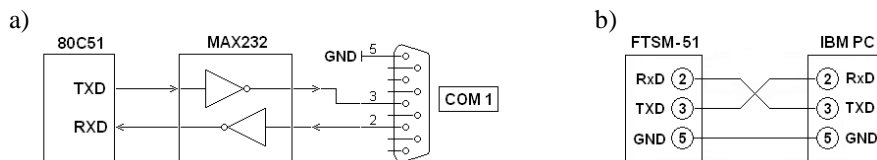
LCD_DANE	- adres początku pola RAM na dane dla LCD;
LCD_FLG	- flagi aktywności zdarzenia (bajt);
LCD_TRYB	- wskaźnik źródła danych (RAM, ROM);

Uwagi dodatkowe:

- Po wprowadzeniu do obszaru pętli programowej procedury automatycznej obsługi wyświetlacza LCD, wyświetlacz przekształca się z urządzenia I/O w pole pamięci RAM (LCD_DANE). Od tego momentu, każde inne zdarzenie, które musi przekazać dane do wyświetlania na wyświetlaczu LCD, wprowadza te dane za pośrednictwem podprogramu LCD_INI - dane do modułu LCD zostaną wyświetlone w sposób automatyczny.
- tabela wzorów wyświetlanych znaków kontrolera HD44780 znajduje się w dodatku (Tabela 1.3.1).

Zadanie 5: obsługa portu szeregowego (łącza RS232).

Mikrokontrolery rodziny MSC-51, które zainstalowane są w zestawach laboratoryjnych posiadają pojedynczy port transmisji szeregowej. Sygnały RxD i TxD portu (końcówki P3.0 i P3.1) są przetwarzane w układzie MAX232 [15] do poziomów napięciowych, charakterystycznych dla łącza RS232. Sygnały te są doprowadzone do złącza CANNON DB9 (złącze typu męskiego) w sposób opisany standardem RS232.



Rys. 1.3.13. Łącze szeregowo RS232: połączenie mikrokontrolera 80C51 ze złączem RS232 (a); połączenie zestawu laboratoryjnego z PC (b).

Sposób połączenie mikrokontrolera ze złączem pokazano na rys.1.3.13a - na rys.1.3.13b pokazano połączenie systemu laboratoryjnego z komputerem, np. IBM_PC.

Opis zadania.

Podprogram obsługi portu transmisji szeregowej, który należy utworzyć w ramach niniejszego zadania - zaleca się, by podprogram nosił nazwę RS_OB-SLUGA. Podprogram powinien umożliwić odbiór i wysyłanie bloku bajtów w trybie asynchronicznym, full duplex¹. Liczba bajtów w bloku odbieranym i nadawanym powinna być ograniczona do 8. Ponieważ zwykle nie jest znany moment odebrania bajtu nadanego przez zewnętrznego nadawcę, w celu uniemożliwienia "zagubienia" bajtu, jego odbiór powinien być przeprowadzony tak

¹ równoległe i niezależne od siebie nadawanie i odbieranie bajtów.

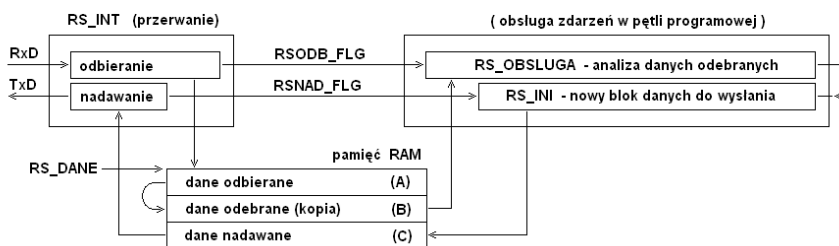
szybko jak się da. W związku z tym, program obsługi zdarzenia powinien być podzielony na 2 segmenty: pierwszy z nich powinien być wykonany w ramach procedury przerwaniowej, a drugi w obszarze pętli programowej.

Dla celów transmisji, w polu wewnętrznej pamięci RAM powinno się zarezerwować ciągły obszar o rozmiarze 24 bajtów (wyjaśnienie w następnym podpunkcie). Początkowy adres tego pola powinien nosić nazwę RS_DANE. Procedura przerwaniowa powinna nosić nazwę RS_INT a pozostała część programu obsługi: RS_OBSLUGA. Zakończenie odbioru bloku bajtów powinno być sygnalizowane przez flagę RSODB_FLG. Zakończenie nadawania bloku bajtów powinno być sygnalizowane przez flagę RSNAD_FLG. Stan aktywny flag powinien być ustawiany przez procedurę RS_INT. Dodatkowo, w celu umożliwienia wysłania bloku danych z dowolnego miejsca wewnętrznej pamięci RAM lub pamięci programu, w ramach omawianego zadania trzeba napisać podprogram pomocniczy o nazwie RS_INI. Podprogram ten powinien skopiować wskazany blok bajtów do właściwego fragmentu pola RS_DANE.

Uwagi o urządzeniach I/O i wykonywaniu zadania.

Odebranie bajtu lub moment zakończenia wysyłania bajtu przez port transmisji szeregowej jest sygnalizowany ustawieniem odpowiedniej flagi. W przypadku odebrania bajtu, flagą ustawianą jest bit RI. W przypadku zakończenia nadawania bajtu - bit TI. Bity RI i TI znajdują się w rejestrze SCON - w rejestrze tym znajdują się inne bity, którymi wybiera się sposób pracy portu (patrz rozdział 1.2.6). Ustawienie któregośkolwiek bitu, RI lub TI, w stan jedynki logicznej może skutkować rozpoczęciem wykonywania przerwania. W ramach programu obsługi przerwania potrzebne jest rozpoznanie przyczyny przerwania (odbior, nadawanie lub oba zdarzenia naraz) i właściwa reakcja na zdarzenie. Ze względu na konieczność obsługi zdarzeń w jak najkrótszym czasie (patrz rozdział 1.4.3), procedura przerwaniowa powinna wykonywać jedynie obsługę rejestru SBUF. W przypadku odebrania bajtu, powinien być on odczytany i umieszczony w pamięci wewnętrznej RAM. W przypadku zakończenia wysyłania bajtu, nowy bajt (jeżeli jest) powinien być odczytany z pamięci wewnętrznej RAM i przesłany do rejestru SBUF. Każde inne czynności, nie związane bezpośrednio z przemieszczaniem danych pomiędzy pamięcią RAM a rejestrem SBUF, powinny być wykonane poza procedurą przerwaniową, np. w obszarze pętli programowej.

Na rys.1.3.14, za pośrednictwem schematu blokowego, pokazano sposób obsługi łącza szeregowego. Na schemacie zaznaczono pamięć RAM oraz 2 bloki działań: procedurę przerwaniową (RS_INT) oraz blok działań pomocniczych, których wykonywanie przekazano do pętli programowej (RS_OBSLUGA, RS_INI). Za pośrednictwem flagi RSNAD_FLG, do pętli programowej jest przesyłana informacja o zakończeniu wysyłania bloku danych. Informacja o zakończeniu odbioru zadeklarowanej liczby bajtów jest przekazywana za pośrednictwem flagi RSODB_FLG.



Rys. 1.3.14. Struktura programu obsługi zdarzeń związanych z odbieraniem i nadawaniem danych przez port transmisji szeregowej.

Pamięć RAM, o rozmiarze 24 bajtów, została podzielona na 3 równe obszary: A, B i C. Początek pola RAM jest wskazywany przez adres **RS_DANE** - jest to również początek pola A, które jest przeznaczone do składowania w nim bieżąco odbieranych bajtów. Początek pola B jest wskazywany przez adres **RS_DANE+8**. Pole B jest przeznaczone do składowania w nim uprzednio odebranego bloku danych (kopia bloku danych odebranych). Początek pola C jest wskazywany przez adres **RS_DANE+16**. Pole C jest przeznaczone do składowania w nim bloku danych, przeznaczonych do nadania.

Procedura przerwaniowa, **RS_INT**, jest wywoływana przez ustawienie flagi **RI** lub **TI** - procedura musi w takim przypadku wykasować ustawioną flagę. Procedura **RS_INT**, w części związanej z nadawaniem bajtów powinna zajmować się wyłącznie pobieraniem kolejnych bajtów z pamięci RAM i wpisywaniem ich do rejestru **SBUF**. Adres bajtu oraz wskaźnik liczby bajtów do wysłania (licznik nadajnika) muszą być przechowywane w pamięci RAM. Po każdym cyklu wysłania bajtu, oba wskaźniki powinny być modyfikowane. Licznik nadajnika powinien być testowany przed wysłaniem bajtu - w przypadku osiągnięcia limitu, zamiast wysłania bajtu powinno ustawić się flagę nadania bloku bajtów, **RSNAD_FLG**. W części związanej z odbiorem bajtów, procedura **RS_INT** powinna zajmować się wyłącznie przepisaniem danej z rejestru **SBUF** do pamięci RAM. Podobnie jak przy nadawaniu, do osiągnięcia tego celu potrzebne jest użycie dwu wskaźników, które przechowują informacje o adresie w polu pamięci RAM i liczbie odebranych bajtów (licznik odbiornika). Po każdym cyklu odbioru bajtu i zapisania go do pamięci RAM, oba wskaźniki powinny być modyfikowane. Licznik odbiornika powinien być testowany po zapisaniu odebranej danej do pamięci RAM - w przypadku osiągnięcia limitu powinno ustawić się flagę odebrania bloku bajtów, **RSODB_FLG**. Jednocześnie, oba wskaźniki, adresowy i licznik odbiornika powinny być ustawione tak, by możliwe było przyjęcie nowego bloku danych. Dzięki takiemu postępowaniu nie jest możliwe przekroczenie pola RAM, które zarezerwowano na dane odbierane. Po przekazaniu do pętli informacji o zakończeniu odbioru bloku danych, w ramach dokończenia obsługi odbioru danych aktywowany jest podprogram **RS_OBSLUGA**, w ramach którego musi być podjęta decyzja o tym, co robić z odebranymi bajtami. W przypadku, gdy analiza odebranego bloku danych należy do programów długich (patrz rozdział 1.4.3, podrozdział "Czas obsługi zdarzenia"), a na-

dawca nie zaprzestał przesyłania danych, może dojść do sytuacji, w której bajty nowego bloku danych mogą być zapisywane w pozycje aktualnie analizowane. Aby uniknąć tego problemu, przed ustawieniem flagi RSODB_FLG, w ramach procedury przerwaniowej powinno dojść do skopiowania odebranego bloku danych z pola A do pola B (patrz rys.1.3.14). Oczywiście, problem ten może być rozwiązany inaczej - po odebraniu bloku danych, wskaźnik adresowy może być przestawiony na początek nowego pola, np. na początek pola B gdy blok odebranych bajtów zgromadzono w polu A i na odwrót. Przy takim rozwiązaniu, do programu RS_OBSLUGA powinna być przekazana dodatkowa informacja o położeniu pola z danymi do analizy.

Jak wspomniano w opisie zadania, w celu umożliwienia prostego sposobu przygotowania danych do wysyłania przez inne zdarzenia, w ramach omawianego zadania trzeba napisać podprogram pomocniczy, RS_INI, umożliwiający rozpoczęcie wysyłania bloku danych z dowolnego miejsca wewnętrznej pamięci RAM lub pamięci programu. Podprogram powinien kopiować wskazany fragment pamięci wewnętrznej RAM lub pamięci programu do pola C pamięci RAM oraz ustawić wskaźnik adresowy nadajnika na początek pola C a do licznika nadajnika wpisać liczbę bajtów do nadania. Przed wywołaniem podprogramu RS_INI, do rejestru ACC powinno się wpisać liczbę bajtów do nadania (1..8), do rejestru B informację o typie pamięci, z której będą kopiowane dane (0-RAM, 1-ROM) a do rejestru DPTR - adres początkowy pola pamięci, które powinno być skopiowane. Uruchomienie procesu nadawania bajtów osiąga się przez ustawienie flagi TI.

I jeszcze jedna uwaga. Wykonywanie programu obsługi zdarzenia przez przerwanie, w każdym przypadku, narzuca konieczność przechowania na stosie tych rejestrów mikrokontrolera, których stan może być naruszony. Mikrokontrolery rodziny MCS-51 umożliwiają bardzo wygodny sposób zachowania stanu 8 rejestrów roboczych, R0..R7. Jest to realizowane przez tzw. przełączanie banków pamięci (patrz rozdział 1.1.1, podrozdział "Pole pamięci danych - RAM") - przez zmianę stanu bitów RS0 i RS1 w rejestrze PSW. W momencie przyjęcia przerwania, należy przełączyć się do banku, który przypisano przerwaniu. W takim przypadku, w trybie natychmiastowym uzyskuje się dostęp do 8 rejestrów, w których może być przechowywana informacja o danych związanych z obsługą przerwania. W omawianym zadaniu, można zapamiętać adres danej odbieranej w rejestrze R0 a adres danej wysyłanej w rejestrze R1. Można również umieścić licznik odbiornika w rejestrze R2 a licznik nadajnika w rejestrze R3. Jeżeli wskazane rejestry będą należały do banku, np. RB1, a bank ten będzie używany przez procedurę RS_INT, to szybki dostęp do tych danych nie będzie stanowił żadnego problemu (a to jest ważne w przypadku procedur przerwaniowych).

Problemy do rozstrzygnięcia:

- poprawne rozdzielenie zadań obsługi zdarzenia pomiędzy przerwanie a program pomocniczy obsługi zdarzenia, wykonywany w pętli programowej;

- sposób zapamiętywania bloku odebranych bajtów - zarządzanie 16-bajtowym polem pamięci wewnętrznej RAM;
- wybór rodzaju transmisji - 8N1, 8E1 lub 8O1 (bez kontroli parzystości lub z włączoną kontrolą nieparzystości albo parzystości) - wybór trybu pracy portu szeregowego;
- wybór źródła taktowania portu transmisji szeregowej i jego właściwa nastawa;
- problem wysyłania większej od 8 liczby bajtów (np. "wizytówka" wysyłana po uruchomieniu sytemu).

Słowa kluczowe:

RS_INT	- nazwa części przerwaniowej podprogramu obsługi;
RS_OBSLUGA	- nazwa części odbiorczej podprogramu obsługi;
RS_INI	- nazwa podprogramu inicjującego nadawanie;
RS_DANE	- adres początku pola RAM na dane odbierane i nadawane.
RSODB_FLG	- flaga zakończenia odbioru grupy bajtów;
RSNAD_FLG	- flaga zakończenia nadania grupy bajtów;

Uwagi dodatkowe:

Po wprowadzeniu do obszaru pętli programowej procedury automatycznej obsługi portu szeregowego, port ten przekształca się z urządzenia I/O w pole pamięci RAM (RS_DANE). Od tego momentu, każde inne zdarzenie, które musi wymienić dane przez port szeregowy, wykorzystuje do tego celu zarezerwowane pole pamięci. W przypadku wysyłania danych, dane są wprowadzane do pola RAM za pośrednictwem podprogramu RS_INI. W przypadku odbioru danych, gotowy do analizy blok danych również znajduje się w pamięci RAM.

Zadanie 6: obsługa łącza I²C.

Łącze opisane nazwą I²C jest cyfrowym łączem typu szeregowego. Standard łącza został opracowany w firmie Philips [16]. Łącze ma bardzo prostą konstrukcję sprzętową i prosty algorytm wymiany informacji. W grupie urządzeń sprzężonych łączem I²C musi być jeden element wyróżniony (ang. master) - element ten nadzoruje pracę łącza. Pozostałe elementy są urządzeniami podrzędnymi (ang. slave). Jest to sytuacja typowa dla systemów mikroprocesorowych gdzie mikroprocesor jest elementem zarządzającym a wszystkie inne urządzenia otoczenia są mu podporządkowane. Każda wymiana informacji pomiędzy urządzeniami systemu odbywa się za pośrednictwem i nadzorem mikroprocesora (mikrokontrolera).

Opis zadania.

W ramach niniejszego zadania trzeba napisać podprogram, który pozwoli na automatyczną wymianę informacji pomiędzy mikrokontrolerem a urządzeniem

peryferyjnym, np. pamięcią EEPROM, przy pomocy łącza I²C. Zaleca się, by podprogram nosił nazwę I2C_OBSLUGA i umożliwiał odbiór lub wysłanie bloku bajtów. Liczba bajtów w bloku odbieranym lub nadawanym powinna być ograniczona do 8. Dla wysyłanych lub odbieranych danych powinno się zarezerwować w pamięci RAM obszar o rozmiarze 10 bajtów - 2 bajty na instrukcje i 8 bajtów na dane. Początkowy adres pola RAM powinien nosić nazwę I2C_DANE. Obszar ten jest wspólny na dane nadawane lub odbierane ponieważ łącze I²C pracuje w systemie half-duplex¹. Rozpoczęcie pracy podprogramu powinno być sygnalizowane przez właściwe ustawienie flagi I2C_FLG - flaga ta powinna być kasowana przez podprogram obsługi w momencie kończenia transmisji danych. Dodatkowo, w celu zainicjowania wymiany informacji przez inne zdarzenia, w ramach omawianego zadania trzeba napisać podprogram pomocniczy o nazwie I2C_INI.

Uwagi o urządzeniach I/O.

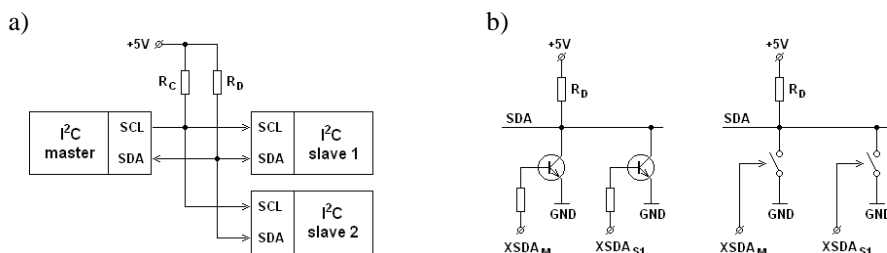
Wymiana informacji pomiędzy mikrokontrolerem a innymi urządzeniami, połączonych łączem I²C, odbywa się za pośrednictwem 2 linii sygnałowych: linii **SCL** (ang. Serial Clock) i linii **SDA** (ang. Serial Data). Linie SCL i SDA tworzą magistralę łącza I²C (rys. 1.3.15a). Końcówki SDA i SCL, są wyprowadzeniami typu "otwarty kolektor" (OC) - połączenie tego typu pokazano na rys. 1.3.15b. Tranzystory końcówek pozwalają na zdefiniowanie zera logicznego linii. Do zdefiniowania stanu jedynki logicznej, w takim przypadku, niezbędne jest zastosowanie tzw. oporników polaryzujących (podciągających, ang. pull-up resistor). Stan jedynki logicznej na linii SDA lub SCL jest osiąganym wtedy, gdy wszystkie tranzystory dołączone do linii są w stanie zablokowania - nie przewożą prądu. Dzięki połączeniu typu OC, można na liniach łącza I²C wymieniać informację w trybie dwukierunkowym. Zakłada się w takim przypadku, że tylko jedno z urządzeń jest w stanie nadawania i tylko ono może włączyć swój tranzystor definiując stan zera logicznego linii - pozostałe są w stanie tzw. nasłuchu i ich tranzystory są zablokowane. O tym, które z urządzeń przejdzie do stanu nadawania decyduje urządzenie nadrzędne - mikrokontroler. Czynność wskazania nadawcy jest realizowana za pośrednictwem protokołu wymiany informacji - protokołu standardu I²C.

W celu uproszczenia opisu, w dalszej części skryptu będą używane skrótowe oznaczenia nazw urządzeń: **M**- urządzenie nadrzędne (MASTER, mikrokontroler) oraz **S** - urządzenie podrzędne (SLAVE).

Każda wymiana informacji przez łącze I²C jest sterowana przez mikrokontroler i jest wykonywana wg tego samego wzoru (formatu), który przedstawiono na rys.1.3.16. Wymiana informacji może być przeprowadzona gdy urządzenia połączone magistralą I²C są w stanie aktywnym. W stanie pasywnym linie SCL i SDA są utrzymywane w stanie jedynki logicznej. O stanie linii SCL zawsze

¹ albo nadawanie, albo odbieranie bloku danych.

decyduje urządzenie nadrzędne¹ - stan linii SDA może być definiowany przez każde z urządzeń. Stan aktywności wszystkich urządzeń jest osiągany w momencie, gdy linia SCL=1 a linia SDA zmienia stan - z jedynki do zera logicznego. Jest to tzw. sekwencja startowa. Przejście w stan pasywny jest osiągalny w momencie gdy linia SCL=1, a linia SDA zmienia stan - z zera do jedynki logicznej. Jest to tzw. sekwencja stopu. Obie sekwencje, startu i stopu, są generowane przez mikrokontroler.



Rys. 1.3.15. Schemat blokowy struktury typowego łącza I²C (a) oraz schemat połączenia typu OC (b).

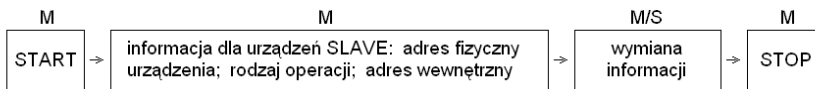
Przesyłanie bajtu odbywa się w 9 cyklach zegarowych, wysyłanych linią SCL. Stan jedynki logicznej na linii SCL oznacza, że bit danej, który jest przekazywany linią SDA jest ustabilizowany - że dana nadaje się do odczytu przez odbiorcę. Zmiana stanu bitu jest możliwa w momencie gdy linia SCL=0. Poszczególne bity są wysyłane w kolejności od najstarszego do najmłodszego, każdy w oddzielnym cyklu zegarowym, numerowanym od 1 do 8. Po wysłaniu ostatniego bitu (bitu D0), nadawca blokuje swój tranzystor wyjściowy - ustawia linię SDA w stan jedynki logicznej. Po odebraniu 8 bitów danej, odbiorca powinien zdecydować, czy odebrana dana jest dla niego przydatna czy też nie. Jeżeli dana została zaakceptowana to odbiorca powinien włączyć swój tranzystor wyjściowy - na linii SDA pojawia się stan zera logicznego. Jest to stan potwierdzenia poprawnego otrzymania informacji. Stan ten jest testowany przez nadawcę w momencie pojawienia się 9 taktu na linii SCL. Stan linii SDA, w takim momencie, jest nazywany bitem potwierdzenia, ACK (ang. acknowledge).



Rys. 1.3.16. Schemat przebiegu transmisji na łączu I²C.

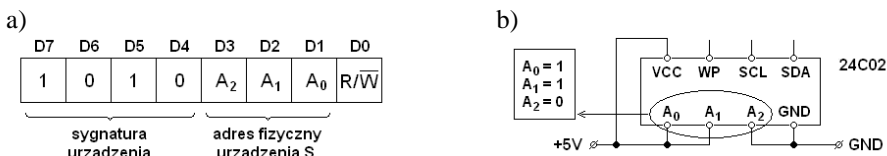
¹ standard dopuszcza przekazanie nadzoru magistrali do każdego urządzenia, które jest do tego przystosowane. Zawsze jednak jest zachowana struktura: jest jedno urządzenie typu MASTER, a reszta pracuje w trybie SLAVE [16].

Opisany wyżej schemat postępowania dotyczy każdego urządzenia dołączonego do magistrali I²C przy czym sygnały START, STOP oraz takty zegarowe na linii SCL są definiowane przez mikrokontroler (M). Mikrokontroler również decyduje o kolejności wysyłania bajtów i ich liczbie. Wymiana informacji odbywa wg schematu pokazanego na rys.1.3.17. Ponieważ jest kilka sposobów przekazywania informacji magistralą I²C, pokazywany dalej będzie charakterystyczny dla szeregowych pamięci EEPROM o pojemności wewnętrznej do 2kB [10]. W takim przypadku, wymiana informacji odbywa się następująco: mikrokontroler aktywuje magistralę sygnałem START i wysyła informację ogólną do wszystkich urządzeń. Informacja ta określa adres fizyczny urządzenia (S) i rodzaj następnej operacji. Urządzenie o wskazanym adresie fizycznym powinno potwierdzić odebranie informacji i odpowiedzieć bitem ACK. Od tego momentu tylko to urządzenie będzie uczestniczyć w dalszej wymianie informacji. Po otrzymaniu potwierdzenia, mikrokontroler wysyła informację o adresie wewnętrznym urządzenia (S) i przystępuje do wymiany danych (zapisuje lub odczytuje dane). Po zakończeniu wymiany informacji, mikrokontroler wysyła sygnał STOP dezaktywując tym samym magistralę I²C.



Rys. 1.3.17. Wymiana informacji łącem I²C.

Jak już wspomniano, po wysłaniu sygnału START, mikrokontroler wysyła bajt z informacją ogólną dla wszystkich urządzeń. Bajt ten jest nazywany bajtem kontrolnym lub statusowym (słowem kontrolnym lub statusowym). W bajcie tym określa się adres fizyczny urządzenia i rodzaj następnej operacji - które z urządzeń, M lub S, będzie wysyłało następny bajt. W przypadku wymiany informacji z pamięcią EEPROM (seria 24CXX, np. 24C02 [10]), po charakterystycznym nagłówku "1010" wysyłany jest adres fizyczny urządzenia i bit oznaczony jako R/W. Gdy R/W=0 to następny bajt będzie wysyłany przez mikrokontroler (operacja zapisu; ang. WRITE); gdy R/W=1 to następny bajt powinien być wysyłany przez pamięć, której adres fizyczny jest zgodny z informacją przesłaną w bajcie kontrolnym (operacja odczytu, ang. READ). Wygląd bajtu kontrolnego oraz sposobu określania adresu fizycznego, pokazano na rys.1.3.18.

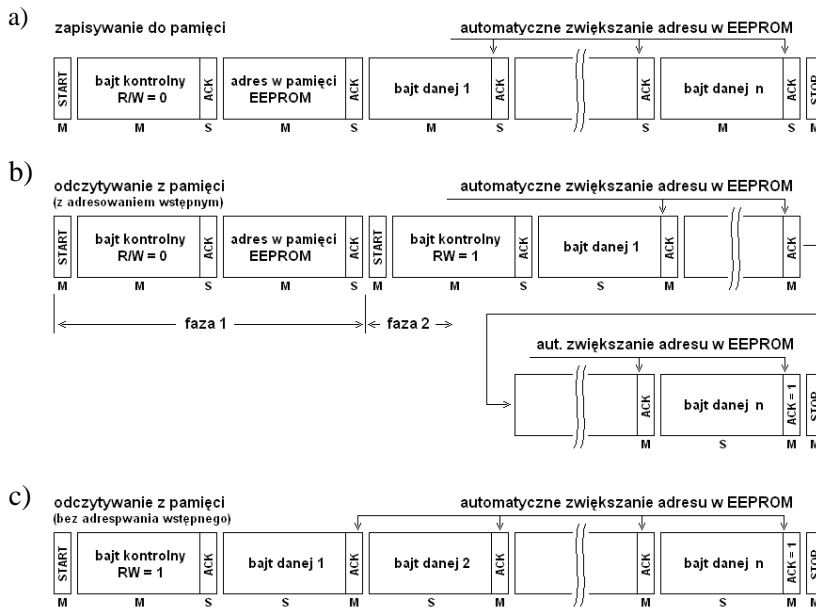


Rys. 1.3.18. Słowo kontrolne dla pamięci EEPROM 24C02 (a) oraz sposób definiowania adresu fizycznego układu scalonego (b) [10].

Na rys.1.3.19. pokazano proces zapisywania i odczytywania danych z pamięci 24C02. Zapisywanie danych powinno odbywać się tak jak to pokazano na rys.1.3.19a. Po wysłaniu sygnału startu i bajtu kontrolnego z bitem R/W=0,

wskazana pamięć 24C02 powinna potwierdzić swoją gotowość do pracy przez wysłanie bitu ACK=0. Po potwierdzeniu, mikrokontroler wysyła bajt adresowy, którego wartość jest wpisywana do wskaźnika adresowego pamięci - jest tym samym określone miejsce, gdzie będzie zapisany bajt danej. Po zaakceptowaniu przez pamięć informacji o adresie (ACK=0), mikrokontroler przesyła bajt lub blok bajtów - po każdorazowym odebraniu bajtu pamięć potwierdza jego otrzymanie przez wysłanie bitu ACK=0. Jednocześnie, w sposób automatyczny, zwiększany jest stan wskaźnika adresowego pamięci o wartość 1. W przypadku pamięci 24C02, liczba bajtów w bloku nie może być większa od 16. Po zakończeniu wysyłania bloku danych, mikrokontroler wysyła sygnał stopu. Po zauważeniu sygnału stopu, pamięć przepisuje odebrane bajty do obszaru pamięci nieulotnej. Proces tego przepisywania danych może trwać nawet kilkanaście ms i w tym czasie pamięć jest niedostępna - na każde wezwanie odpowiada bitem ACK=1.

Odczytywanie danych z pamięci nie jest limitowane - można odczytywać dowolną liczbę bajtów. Odczytywanie można prowadzić dwoma sposobami. Pierwszy z nich polega na tym, że przed odczytaniem bajtu lub bloku bajtów, mikrokontroler musi przekazać pamięci stan wskaźnika adresowego - określić adres bajtu do odczytu. Drugi sposób polega na bezpośrednim odczytywaniu bajtów wg wskazań wskaźnika adresowego pamięci, który był ustawiony poprzednimi procesami zapisu lub odczytu. Obie sytuacje przedstawiono na rys. 1.3.19b i rys.1.3.19c.



Rys. 1.3.19. Zapisywanie i odczytywanie danych z pamięci EEPROM 24C02.

Jak widać na rysunku rys.1.3.19b, w pierwszej fazie odczytywania informacji z pamięci wykonywane są czynności identyczne jak w przypadku procedury zapisu - po sygnale startu, do pamięci jest przekazywana informacja o stanie wskaźnika adresów. Po zdefiniowaniu adresu wewnętrznego w pamięci RAM, mikrokontroler przechodzi do fazy drugiej procesu odczytywania i ponownie generuje sygnał startu oraz wysyła bajt kontrolny z bitem R/W=1. Takie ustawienie bitu oznacza, że od tego momentu, mikrokontroler przechodzi w stan nasłuchu - będzie odbierał bity danej wysyłanej przez pamięć. Po każdym wysłaniu przez pamięć bajtu danej, wskaźnik adresu pamięci jest zwiększany o 1. Po każdym odebraniu danej, mikrokontroler wysyła bit ACK=0 w przypadku, gdy chce odbierać dalsze bajty lub ACK=1, gdy skończył odbiór wymaganej liczby danych. Po wysłaniu bitu ACK=1, mikrokontroler wysyła sygnał stopu - dezaktywuje łącze I²C.

W przypadku pokazanym na rys.1.3.19c, mikrokontroler od razu przechodzi do fazy drugiej opisanego wyżej procesu - dane z pamięci są odczytywane po wysłaniu przez mikrokontroler sygnału startu i bajtu kontrolnego z bitem R/W=1.

Uwagi o wykonywaniu zadania.

System mikroprocesorowy FTSM_51 umożliwia wskazanie końcówek mikrokontrolera, które będą stanowić łącze I²C (SCL i SDA). Wybór może być wykonywany za pośrednictwem okna nastaw w programie FTT_MONITOR. Zakłada się, że w przypadku braku modułu rozszerzenia, urządzeniem docelowym będzie pamięć wirtualna EEPROM o rozmiarze do 2048 bajtów z protokołem transmisyjnym pamięci serii 24CXX [10]. Wybór typu pamięci jest możliwy za pośrednictwem okna nastaw w programie FTT_MONITOR. W przypadku dołączenia do systemu FTSM_51 modułu rozszerzenia, dane w oknie nastaw mogą ulec modyfikacji i będą odzwierciedlały architekturę dołączonego urządzenia.

Patrząc na rys.1.3.19, można zauważyć, że w przypadku zapisywania danych do pamięci EEPROM, mikrokontroler wykonuje działania związane wyłącznie z nadawaniem bajtów. Wszystkie te działania są identyczne ze sobą. Nie ma różnicy pomiędzy nadawaniem bajtu kontrolnego, adresu czy też bajtu danej - różna jest jedynie treść wysyłanej informacji. Identyczna sytuacja występuje w przypadku odbierania danych. Zaleca się zatem napisanie 2 modułów programowych, z których pierwszy będzie powiązany z nadawaniem a drugi z odbieraniem bajtu - w takim przypadku, działanie podprogramu I2C_OBSLUGA będzie sprowadzać się do właściwego zarządzania kolejnym wywoływaniem modułów. Zaleca się również, by w ramach pojedynczego cyklu obsługi pętli programowej był odbierany lub wysyłany nie więcej niż jeden bajt danej.

Podprogram I2C_INI powinien umożliwić rozpoczęcie obsługi łącza I²C. Podprogram I2C_OBSLUGA powinien otrzymać informację o trybie pracy, o adresie źródła danych i liczbie przesyłanych bajtów. Zaleca się, by miejsce

danych w pamięci RAM było stałe i początkowy adres tego miejsca był określony etykietą I2C_DANE. Informacja o liczbie bajtów do przesłania powinna być przekazana przez rejestr ACC; przez rejestr B powinno się wprowadzić informację o numerze trybu pracy a przez DPL - informację o adresie źródła danych do przesłania. Dobór rejestrów pomocniczych dla podprogramu I2C_OBSLUGA i ich lokalizacja w pamięci RAM pozostawiono decyzji studenta.

Podprogram I2C_INI powinien umożliwić rozpoczęcie obsługi łącza I²C. Podprogram obsługi, I2C_OBSLUGA, powinien otrzymać informację o trybie pracy, o adresie źródła danych i liczbie przesyłanych bajtów. Zaleca się, by miejsce danych w pamięci RAM było stałe (wydzielone pole RAM) i początkowy adres tego miejsca był określony etykietą I2C_DANE. Informacja o liczbie bajtów do przesłania powinna być przekazana przez rejestr ACC; przez rejestr B powinno się wprowadzić informację o numerze trybu pracy, a przez DPL - informację o adresie źródła danych do przesłania. Dobór rejestrów pomocniczych dla podprogramu I2C_OBSLUGA i ich lokalizację w pamięci RAM pozostawiono decyzji studenta.

Problemy do rozstrzygnięcia.

- organizacja programu I2C_OBSLUGA;
- reakcja na pojawienie się błędu (ACK=1);

Słowa kluczowe:

I2C_OBSLUGA	- nazwa podprogramu obsługi łącza I ² C;
I2C_INI	- nazwa podprogramu inicjującego pracę łącza;
I2C_DANE	- adres początku pola RAM na dane;
I2C_FLG	- flaga aktywności zdarzenia;

Uwagi dodatkowe:

Po wprowadzeniu do obszaru pętli programowej procedury automatycznej obsługi łącza I²C, port ten przekształca się z urządzenia I/O do pola pamięci RAM (I2C_DANE). Od tego momentu, każde inne zdarzenie, które musi wymienić dane przez łącze I²C, wykorzystuje do tego celu zarezerwowane pole pamięci. W przypadku wysyłania danych, dane są wprowadzane do pola RAM za pośrednictwem podprogramu I2C_INI. W przypadku odbioru danych, gotowy do analizy blok danych również znajduje się w pamięci RAM (I2C_DANE).

Zadanie 7: obsługa enkodera obrotowego.

Enkoder obrotowy jest cyfrowym odpowiednikiem analogowego elementu wprowadzania nastawy, np. potencjometru siły głosu w radioodbiorniku. Bardzo dobrym przykładem budowy i działania enkodera jest konstrukcja i działanie tradycyjnej myszki komputerowej, w której przemieszczenie myszki było zamieniane na informację cyfrową. Bardzo często enkoder obrotowy jest nazywany przetwornikiem obrotowo-impulsowym lub pulsatorem.

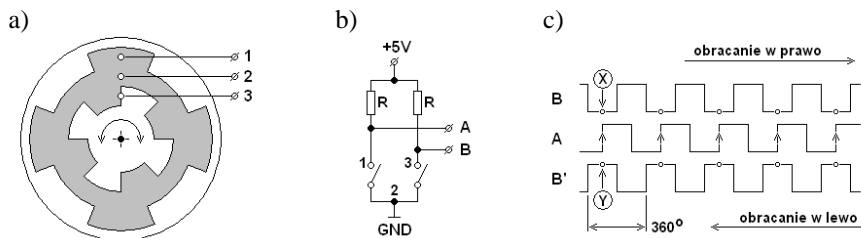
Opis zadania.

W ramach niniejszego zadania trzeba napisać podprogram, który pozwoli na automatyczne wprowadzanie do systemu mikroprocesorowego informacji o ruchu obrotowym wału enkodera. Zaleca się, by podprogram nosił nazwę ENK_OBSLUGA. Podprogram powinien rozpoznać fakt zmiany położenia wału enkodera i wprowadzić wielkość przemieszczenia do rejestru o nazwie ENK_STAN. Wielkość przemieszczenia powinna być wprowadzona w kodzie U2 (obrót wału w lewo lub w prawo). Każda zmiana stanu rejestru powinna być sygnalizowana za pośrednictwem flagi ENK_FLG. Podprogram ENK_OBSLUGA powinien zapewnić eliminację problemu pseudobrotu wału enkodera i zmieniać stan flagi ENK_FLG oraz rejestru ENK_STAN jedynie w przypadku jednoznacznego pozyskania informacji o stanie enkodera.

Uwagi o urządzeniach I/O i wykonywaniu zadania.

System mikroprocesorowy FTSM_51 jest wyposażony w wirtualny enkoder obrotowy. Za pośrednictwem okna nastaw programu FTT_MONITOR można wskazać końcówki mikrokontrolera, które będą stykami enkodera. W polu okna nastaw można również określić liczbę pól stykowych enkodera przypadających na 1 obrót wału. W przypadku dołączenia do systemu FTSM_51 modułu rozszerzenia z rzeczywistym enkoderem, dane w oknie nastaw mogą ulec modyfikacji i będą odzwierciedlały architekturę dołączonego urządzenia.

Na rys.1.3.20a pokazano wygląd tarczy kodowej enkodera 4-polowego oraz sposób włączenia enkodera do obwodu elektrycznego (rys.1.3.20b). Styki elektryczne (1,2 i 3) dotykają do tarczy kodowej urządzenia, której zacieniowany obszar przewodzi prąd elektryczny. Obracanie wałem enkodera powoduje zwieranie styków 1 i 3 ze stykiem 2 co umożliwia otrzymanie sygnałów elektrycznych A i B, pokazanych na rys.1.3.20c. Geometria pola stykowego powoduje, że sygnały te są przesunięte względem siebie. Przy obracaniu tarczy kodowej w prawo, sygnał B jest opóźniony względem A o kąt względny 90° (kąt względny 360° odpowiada pełnemu okresowi sygnału A lub B). Przy obracaniu tarczy kodowej w lewo, sygnał B wyprzedza A o kąt względny 90° . Można zauważyć, że obserwując stan sygnału B w momencie zmiany stanu sygnału A, np. z 0 na 1, uzyskuje się informację o kierunku obrotu wału enkodera (rys. 1.3.20c: punkty X i Y sygnału B).



Rys. 1.3.20. Tarcza kodowa enkodera (a); sposób włączenia enkodera do obwodu elektrycznego (b) i sygnały generowane przez enkoder (c).

W przypadku posługiwania się enkoderami istnieje problem podobny do problemu wibracji styków, który opisano w zadaniu 2. W odróżnieniu od poprzednio opisywanego, w przypadku enkodera problem jest związany przede wszystkim z mikroprzemieszczeniami wału enkodera w pobliżu granicy styku - jeżeli wał enkodera zostanie pozostawiony w takim położeniu, to nawet drobne drgania konstrukcji mogą powodować pojawienie się informacji o zwarciu lub rozwarciu styków - pseudoinformacji o obrocie wału (informacji o pseudoobrocie). Problem daje się usunąć metodami podobnymi do zastosowanych w zadaniu 2.

Problemy do rozstrzygnięcia:

- określenie optymalnego czasu wywoływania podprogramu ENK_OBSLUGA;
- likwidacja problemu pseudoobrotu wału enkodera;
- zwiększenie rozdzielczości enkodera (2x).

Słowa kluczowe:

ENK_OBSLUGA	- nazwa podprogramu obsługi enkodera;
ENK_STAN	- bufor na wartość zmiany pozycji wału enkodera;
ENK_FLG	- flaga wykrycia ruchu wału enkodera

Uwagi dodatkowe:

- do wykonania zadania należy się posłużyć modułem T0_OBSLUGA (patrz zadanie 1), którego wybrana flaga wskaże właściwy moment wywołania podprogramu ENK_OBSLUGA;
- po wprowadzeniu do obszaru pętli programowej procedury automatycznej obsługi enkodera, przekształca się on z urządzenia I/O w rejestr pamięci RAM, ENK_STAN. Od tego momentu, każde inne zdarzenie, które chce pozyskać informację o stanie enkodera, może to osiągnąć po zaobserwowaniu aktywnego stanu flagi ENK_FLG.

Zadanie 8: obsługa silnika krokowego.

Silnik krokowy jest silnikiem elektrycznym, którego wał może się ustawiać w ściśle określonych pozycjach, determinowanych konstrukcją mechaniczno-elektryczną silnika. Obrót wału silnika jest realizowany przez kolejne przejścia do sąsiedniej pozycji - mówi się w takim przypadku o wykonaniu kroku. W zależności od typu silnika, pełny obrót wału następuje po wykonaniu N kroków (np. 200 kroków w przypadku silników stosowanych w stacjach dyskiety).

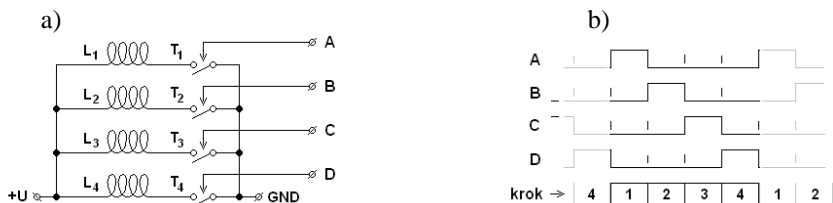
Opis zadania.

W ramach niniejszego zadania trzeba napisać podprogram, który pozwoli na automatyczne sterowanie pracą silnika krokowego. Zaleca się, by podprogram nosił nazwę SK_OBSLUGA. Podprogram powinien umożliwić wykonanie zadanej liczby kroków i obrócić wał silnika w lewo lub prawo - po każdorazowym wywołaniu podprogramu powinien być wykonany pojedynczy krok. Liczba

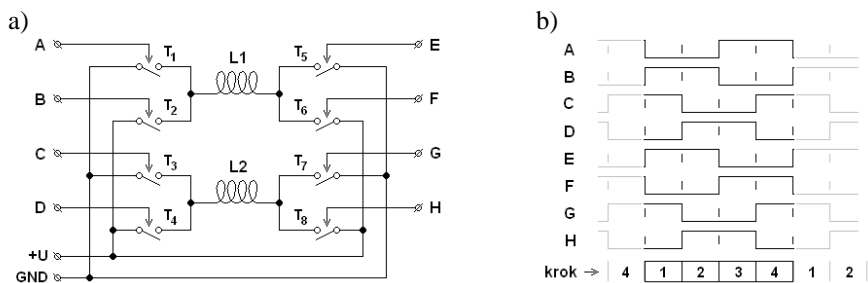
kroków do wykonania powinna być przechowywana w rejestrze o nazwie SK_KROKI. Liczba kroków powinna być liczbą w kodzie U2 (obrót wału w lewo lub w prawo). Regulowana powinna być również szybkość obracania wału. Czas pomiędzy wykonaniem kolejnych kroków powinien być przechowywany w rejestrze o nazwie SK_CZAS. Podprogram SK_OBSLUGA powinien zapamiętywać sumaryczną liczbę kroków wykonanych od umownej chwili - np. od momentu ustawienia wału w określonej pozycji. Informacja ta powinna być przechowywana w 2-bajtowym rejestrze o nazwie SK_STAN. Stan aktywności podprogramu SK_OBSLUGA powinien być sygnalizowany flagą SK_FLG. Dodatkowo, w celu zainicjowania pracy silnika, w ramach omawianego zadania trzeba napisać podprogram pomocniczy o nazwie SK_INI.

Uwagi o urządzeniach I/O i wykonywaniu zadania.

System mikroprocesorowy FTSM_51 jest wyposażony w wirtualny silnik krokowy. Stan silnika jest określany przez stan bajtu, który jest wysyłany pod adres określony za pośrednictwem okna nastaw programu FTT_MONITOR. Przyjęto zasadę, że bity bajtu włączają tranzystory sterujące cewkami silnika wtedy, gdy ich wartość jest jedynką logiczną. Pokazane na rysunkach 1.3.21 i 1.3.22 bity sterowania, oznaczone literami od A do H, pokrywają się z bitami bajtu sterowania, odpowiednio od D₀ do D₇. W polu okna nastaw można również określić typ silnika oraz liczbę kroków przypadających na pełny obrót wału silnika. W przypadku dołączenia do systemu FTSM_51 modułu rozszerzenia z rzeczywistym silnikiem, dane w oknie nastaw mogą ulec modyfikacji i będą odzwierciedlały architekturę dołączonego urządzenia.



Rys. 1.3.21. Zasilanie silnika czterofazowego: schemat połączeń (a) i przebiegi sterowania pracą silnika (b).



Rys. 1.3.22. Zasilanie silnika dwufazowego: schemat połączeń (a) i przebiegi sterowania pracą silnika (b).

Działanie silników krokowych polega na wytworzeniu wewnątrz silnika pola magnetycznego, które ustawia namagnesowany rotor silnika w pozycji najbardziej korzystnej z punktu widzenia oddziaływania wzajemnego 2 pól magnetycznych. Pole magnetyczne jest wytwarzane przez zespół cewek, które są umieszczone w rdzeniu stojana silnika, w specjalny sposób. Najbardziej popularne silniki krokowe są budowane jako struktury dwu lub czterocewkowe i nazywane są, odpowiednio, silnikami dwufazowymi i czterofazowymi. W silnikach dwufazowych znajdują się 2 cewki przez które, w każdym momencie, przepływa prąd elektryczny. Ustawienie wału silnika jest determinowane, z elektrycznego punktu widzenia, przez kierunki prądu wprowadzonego do poszczególnych cewek silnika. W silnikach czterofazowych znajdują się 4 cewki a prąd przepływa przez jedną z nich. Ustawienie wału silnika jest determinowane przez cewkę aktywną - tę, przez którą przepływa prąd elektryczny. W każdym przypadku, właściwe zasilanie cewek umożliwi ustawienie się wału silnika w 4 pozycjach. Inne pozycje są powtórzeniem opisanych. Sposób zasilania silników pokazany jest rysunkach 1.3.21 i 1.3.22. Elementami przełącznikowymi są tranzystory, od T_1 do T_8 . Poprawna praca silnika jest możliwa w przypadku gdy zmiana warunków zasilania silnika spowoduje przemieszczenie rotora do sąsiedniej pozycji - w lewo lub prawo. Każde inne działanie może spowodować przejście rotora do pozycji niekontrolowanej.

W przypadku zasilania silnika dwufazowego istnieje możliwość popełnienia błędu i zwarcia obwodu zasilania. Dla układu rzeczywistego mogłoby to spowodować zniszczenie zasilacza lub tranzystorów przełączających. W każdym przypadku, zarówno przy pracy z obiektem rzeczywistym czy też pracy w trybie wirtualnym, fakt zwarcia zostanie rozpoznany przez system FTSM_51. System przerwie pracę programu użytkownika a program FTT_MONITOR wygeneruje stosowny komunikat tekstowy.

Automatyczna obsługa silnika polega na wpisywaniu bajtu sterowania podabrany adres w przestrzeni adresowej urządzeń I/O. Za obsługę silnika powinien odpowiadać podprogram SK_OBSLUGA, który musi posiadać informację o liczbie kroków do wykonania. System czasowo-licznikowy powinien otrzymać informację o częstotliwości wywoływania podprogramu SK_OBSLUGA. Do wprowadzenia tych danych powinno używać się podprogramu SK_INI. Przed wywołaniem programu powinno się mu przekazać właściwe dane: informacja o liczbie kroków do wykonania powinna być przekazana przez rejestr ACC; przez rejestr B powinno się wprowadzić informację o odstępach czasowych pomiędzy wykonaniem kolejnych kroków silnika. Dodatkowo, podprogram SK_INI powinien umożliwić skasowanie informacji o bieżącym stanie silnika (w umownej chwili T_0). Sposób wykonania tej operacji pozostawiono decyzji studenta.

Problemy do rozstrzygnięcia:

- regulacja szybkości obrotowej silnika (częstotliwość wywoływań podprogramu SK_OBSLUGA);

- ustalenie zestawu bitów sterowania dla konkretnego typu silnika.

Słowa kluczowe:

SK_OBSLUGA	- nazwa podprogramu obsługi silnika;
SK_INI	- nazwa podprogramu inicjującego pracę silnika;
SK_STAN	- bufor na bieżący stan silnika (od umownej chwili T_0);
SK_KROKI	- licznik liczby kroków do wykonania;
SK_CZAS	- bufor na stałą czasową wykonywania kolejnych kroków silnika;
SK_FLG	- flaga aktywności podprogramu obsługi silnika.

Uwagi dodatkowe.:

- do wykonania zadania należy się posłużyć zmodyfikowanym (w ramach bieżącego zadania) modułem T0_OBSLUGA, którego wybrana flaga wskazuje właściwy moment wywołania podprogramu SK_OBSLUGA;
- po wprowadzeniu do obszaru pętli programowej procedury automatycznej obsługi silnika, przekształca się on z urządzenia I/O w kilka rejestrów pamięci RAM: SK_STAN, SK_KROKI i SK_CZAS. Od tego momentu, każde inne zdarzenie, które chce pozyskać informację o stanie silnika, może to osiągnąć po odczytaniu rejestru SK_STAN; każde inne zdarzenie może rozpocząć pracę silnika po uruchomieniu programu SK_INI.

Zadanie 9: zegar czasu rzeczywistego.

Opis zadania.

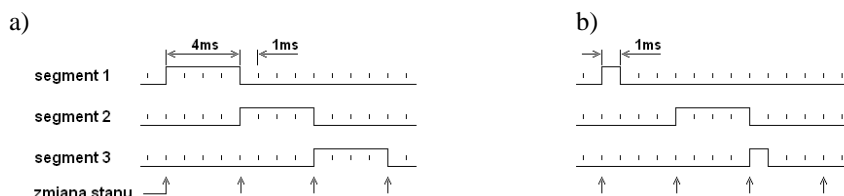
W ramach niniejszego zadania trzeba napisać podprogram o nazwie ZEGAREK, który przekształci system FTSM_51 w 24-godzinny zegar czasu rzeczywistego. Podprogram powinien być umieszczony w pętli programowej i powinien być aktywowany co 1s. Powinien on umożliwić wskazywanie bieżącego czasu na wielosegmentowym wyświetlaczu LED w formacie: "GG.MM.SS" (GG - godzina, MM - minuta, SS - sekunda).

Przekształcony do postaci zegarka system FTSM_51 powinien, tak jak każdy zegarek, dać się regulować. Do tego celu powinno się użyć klawiatury systemu. Podprogram sterowania nastawami, wykorzystujący dane z klawiatury, powinien nosić nazwę Z_NASTAWA. W ramach działania tego programu, po zatrzymaniu zegara klawiszem, np. "C" (nr 20), za pośrednictwem przycisków przesuwania w poziomie (nr 1 lub 2), należy wskazać co ma być ustawiane: godziny, minuty czy sekundy. Po tym ustawieniu, za pośrednictwem przycisków przesuwania w pionie (nr 0 lub 3), należy ustawić pożądaną wartość czasu. Ponowne uruchomienie zegarka może nastąpić, np. po naciśnięciu przycisku "=" (nr 23).

Uwagi o urządzeniach I/O i wykonywaniu zadania.

Do wykonania zadania należy posłużyć się procedurami: T0_OBSLUGA

(zadanie 1), KBD_OBSLUGA (zadanie 2) oraz LED_OBSLUGA (zadanie 3). Procedury te zapewnią automatyczną obsługę odmierzania czasu w systemie, obsługę klawiatury i obsługę wyświetlacza LED.



Rys. 1.3.23. Regulacja jasności świecenia wskaźników LED za pomocą zmiany czasu aktywności segmentów wyświetlacza.

Procedura obsługi wyświetlacza LED (LED_OBSLUGA) umożliwia automatyczne wyświetlanie danych z jednakową jasnością. To może jednak być przeszkodą w przypadku próby nastawiania zegarka. Jednakowa jasność świecenia elementów wyświetlacza nie pozwala na wskazanie danych do nastawy (godziny, minuty lub sekundy). Optymalną sytuacją byłoby to, by wyróżniony fragment wyświetlacza jest jaśniejszy (ciemniejszy) od pozostałej części. Opisany efekt można osiągnąć przez regulację czasu świecenia poszczególnych segmentów wyświetlacza - pokazano to na rys.1.3.23. Zakładając, że w czasie pracy "normalnej" wszystkim segmentom przydzielono jednakowy czas, np. 4ms, to skrócenie tego czasu do, np. 1ms spowoduje, że segmenty ze skróconym czasem aktywności będą wyświetlały daną ze zmniejszoną intensywnością.

Problemy do rozstrzygnięcia:

- zmiana jasności świecenia wybranych fragmentów wyświetlacza LED;
- umiejscowienie w pętli programowej i sposób aktywowania podprogramu Z_NASTAWA

Słowa kluczowe:

ZEGAREK - nazwa podprogramu odmierzania czasu;
Z_NASTAWA - nazwa podprogramu ustawiania czasu;

Uwagi dodatkowe:

- do wykonania zadania należy zmodyfikować (w ramach bieżącego zadania) moduł LED_OBSLUGA.

Zadanie 10: zabezpieczanie działania systemu układem WDT.

Bardzo często, jedynym sposobem ratunku przed sytuacją nieprzewidywalną, np. zawieszeniem się pracy mikrokontrolera, jest natychmiastowe jego wyłączenie i ponowne włączenie. Operację tę można wykonać za pomocą sprzętowego sygnału kasowania mikrokontrolera, który może być wytworzony ręcznie lub automatycznie. Do automatycznego kasowania mikrokontrolera, szczególnie dobrze nadaje się układ WDT (ang. watchdog timer).

Opis zadania.

W ramach niniejszego zadania trzeba napisać podprogram, który zabezpieczy działanie systemu mikroprocesorowego przed przypadkowym zawieszeniem jego pracy. Oczywiście, w przypadku tego zadania, pojęcie *przypadkowe zawieszenie* musi być spowodowane w sposób kontrolowany przez utworzenie, np. pętli pustej w obszarze pętli programowej.

Wykonanie zadania polega udowodnieniu, że po zaistnieniu problemu z obserwacją stanu środowiska można go usunąć przez zastosowanie licznika WDT. Wszystkie elementy zadania, tzn. wprowadzenie do pętli programowej kontrolowanego zaburzenia, wizualizacja tego zdarzenia oraz wizualizacja ochrony systemu przez licznik układu alarmowego powinny być autorskim pomysłem studenta.

Uwagi dodatkowe:

- materiałem pomocniczym jest opis układu WDT mikrokontrolera 89S8253, który przedstawiono w rozdziale 1.2.8.
- w przypadku systemu DSM-51, działanie układu WDT opisano w podręczniku systemu [3].

CZEŚĆ 2

O PROGRAMOWANIU MIKROKOMPUTERÓW

2.1. ŚRODOWISKO MIKROKOMPUTERA I JEGO OBSŁUGA.	100
2.1.1. URZĄDZENIA I ZDARZENIA W ŚRODOWISKU KOMPUTERA.	100
2.1.2. PRZYGOTOWANIE ŚRODOWISKA I PĘTLA PROGRAMOWA.	102
2.1.3. OBSŁUGA ZDARZEŃ W PĘTLI PROGRAMOWEJ.	104
2.2. PROGRAMOWANIE MIKROKOMPUTERÓW.	121
2.2.1. PROGRAMOWANIE W JĘZYKU ASEMBLERA.	121
2.2.2. PROGRAMOWANIE W JĘZYKU C.	127

1.4. Środowisko mikrokomputera i jego obsługa.

1.4.1. Urządzenia i zdarzenia w środowisku komputera.

Przed rozpoczęciem jakichkolwiek rozważań związanych ze strukturą oprogramowania minikomputerów warto zdefiniować kilka pojęć, charakterystycznych dla systemów mikroprocesorowych, zarówno tych małych jak i dużych.

Jak już wspomniano, system mikroprocesorowy składa się z jednostki centralnej CPU (mikroprocesora), układów elektronicznych pamięci oraz wszystkich innych układów, którym przydzielono ogólną nazwę *urządzeń wejścia/wyjścia* (I/O, ang. input/output) lub krócej: urządzeń.



urządzenie I/O to każdy fizyczny element komputera podlegający kontroli i sterowaniu przez jednostkę centralną - mikroprocesor ..

Urządzenia I/O są budowane pod kątem spełnienia określonych zadań w systemie mikroprocesorowym, np. obsługi transmisji szeregowej, transmisji równoległej, generacji obrazu monitora itp. Przejmują one dużą część zadań mikroprocesora, pozwalając tym samym na odciążenie mikroprocesora od wypełniania zadań bardzo specyficznych - mikroprocesor pozyskuje dodatkowy czas na wykonywanie innych zadań. Konstrukcja urządzeń I/O jest często bardzo zaawansowana a sposób działania zależy od wstępnie wprowadzonych informacji do ich rejestrów kontrolnych. Mówimy w takim przypadku o możliwości programowania urządzeń I/O.

Zespół układów scalonych I/O, pomagający mikroprocesorowi wykonywać określone zadania, określa się mianem środowiska sprzętowego mikroprocesora lub krócej: środowiska.



środowisko sprzętowe to grupa urządzeń, których obecność jest niezbędna do zapewnienia poprawnej pracy komputera ..

Podstawowym i jedynym zadaniem mikroprocesora jest wykonywanie programu będącego zestawem rozkazów umieszczonych w pamięci komputera. Wymieniając program można spowodować zupełnie inną pracę elementów komputera, pomimo tego, że same elementy komputera pozostały bez zmian. Ponieważ, zarówno środowisko sprzętowe jak i oprogramowanie komputera nie mogą stanowić bytów niezależnych, można uogólnić określenie środowiska komputera, rozszerzając je na sprzęt i oprogramowanie (ang. hardware and software).



środowisko to grupa urządzeń i program, których obecność jest niezbędna do zapewnienia poprawnej pracy komputera ..

Każdy zauważył, co się dzieje z komputerem po włączeniu zasilania. Na monitorze pojawiają się różne komunikaty, informujące o postępie przeobrażeń stanu komputera. Po zakończeniu fazy uruchamiania, na ekranie monitora pozostaje plansza końcowa, a na niej, np. kursor myszki. I co dalej? Dalej NIC - nic się nie dzieje. Ale mikroprocesor komputera nie zatrzymał się, on dalej pracuje i wykonuje kolejne rozkazy. Wystarczy jednak przesunąć myszkę komputera, by na ekranie zauważyć przesunięcie jej kursora - a później znowu NIC. Co się właściwie dzieje?

Badając inne elementy komputera, np. przyciski myszki, klawiaturę komputera itp. można zauważyć, że w każdym przypadku, działanie komputera jest praktycznie identyczne: jeżeli coś się stało to następuje reakcja na to COŚ! Ponieważ mikroprocesor w sposób ciągły wykonuje rozkazy, powyższe działanie można opisać następująco: mikroprocesor za pośrednictwem swojego oprogramowania w sposób ciągły testuje stan środowiska - po rozpoznaniu zmiany stanu jednego z urządzeń, mikroprocesor wykonuje dodatkowy program, który jest przypisany temu urządzeniu. Mówimy w takim przypadku o pojawieniu się zdarzenia i reakcji na to zdarzenie (obsługa zdarzenia).



zdarzenie to każda zmiana stanu środowiska - zwykle jest to zjawisko odstępstwa od stanu uznawanego za stan równowagi (stan pasywny) ..



obsługa zdarzenia to wykonywanie programu z dedykowanej listy instrukcji (podprogramu), pozwalającego zareagować na fakt zaistnienia zdarzenia ..

Z określeń podanych wyżej wynika, że fakt wystąpienia zdarzenia pociąga za sobą obsługę tego zdarzenia. Oczywiście, musi istnieć prosty mechanizm rozpoznawania wystąpienia zdarzenia. Najprostszym sposobem osiągnięcia tego celu jest przypisanie każdemu zdarzeniu elementu wskaźnikowego, tzw. flagi. Flagą może być bit lub bajt pamięci RAM albo rejestr urządzenia I/O - z punktu widzenia mikroprocesora jest to zupełnie obojętne ponieważ odwołanie do tych elementów jest identyczne: jest nim odczyt danej.



flaga to wskaźnik określający stan elementu środowiska (zdarzenia) - stan aktywny flagi oznacza pojawienie się zdarzenia ..

Każdej fladze powinno się przypisać dwa stany: stan pasywny i stan aktywny. Stan pasywny flagi oznacza brak wystąpienia zdarzenia.

Każda flaga, dowolnego zdarzenia, powinna mieć precyzyjnie określony stan pasywny - może to być stan logicznego zera w przypadku bitu lub wartość 0 w przypadku bajtu. Każdy inny stan flagi jest uznawany za stan aktywny i oznacza wystąpienie zdarzenia. Można zauważyć, że gdy flagą jest bit, to jedyną informacją stanu aktywnego jest to, że "coś się stało". Jeżeli flagą jest, np. bajt, to oprócz informacji, że "coś się stało", można dodatkowo przekazać informację "co się stało".

Rozważane do tej pory pojęcie zdarzenia było powiązane z pojęciem zdarzenia sprzętowego i przypisanego temu zdarzeniu programowi obsługi zdarzenia. Można jednak zauważyć, że działanie typowo programowe też posiada cechy zdarzenia. Jeżeli przyjąć, że zdarzeniem jest zlecenie rozpoczęcia obliczeń to reakcją na to zdarzenie jest wykonanie obliczeń i podanie wyniku. Podobnie, jak w przypadku typowych urządzeń I/O, rejestr lub bit, który niesie informację o trwaniu lub zakończeniu obliczeń, można nazwać flagą. Ponieważ nie tylko sprzęt ale i zadania typowo obliczeniowe stanowią o istocie komputera, można spróbować zmodyfikować definicję środowiska. Podana wcześniej definicja zdarzenia nie zmienia się.



środowisko to grupa zdarzeń, których obsługa jest niezbędna do zapewnienia poprawnej pracy komputera ..

1.4.2. Przygotowanie środowiska i pętla programowa.

Z przeprowadzonych wyżej rozważań wynika, że pracę komputera, w sposób ogólny, można sprowadzić do wykonywania dwu czynności: obserwacji stanu środowiska i obsługi zdarzeń. Obie czynności są wykonywane przez program komputera. Ponieważ pojemność pamięci programu komputera jest ograniczona, jest rzeczą oczywistą, że w pewnym momencie musi nastąpić powrót do czynności już wykonywanych i rozpoczęcie ponownego ich wykonywania. Takie powtarzanie działania, już wykonanego, nosi nazwę pętli programowej. Program pętli programowej realizuje podstawowe zadanie komputera: obserwację środowiska i obsługę zdarzeń.



pętla programowa to powtarzany cyklicznie fragment programu, w którym wykonywana jest obsługa zdarzeń ..

Przed rozpoczęciem pracy w pętli, wszystkie urządzenia I/O powinny być przygotowane do pracy. Urządzenia wymagające ustalenia trybu pracy powinny zostać zaprogramowane przed ich użyciem przez wpisanie właściwej informacji do ich rejestrów kontrolnych. Jeżeli urządzenia posiadają wyjściowe linie sygnałowe, to stan logiczny tych linii powinien być również określony i dostosowany do wymogów środowiska. Dokonuje się tego poprzez wprowadzenie

informacji do innych rejestrów urządzenia, nazywanych rejestrami danych. Przygotowaniu wstępnemu powinien podlegać również sam mikroprocesor, np. powinien być określony sposób przyjmowania przerw oraz, jeżeli jest to konieczne, ustawieniu wstępnemu powinny podlegać wybrane rejestry mikroprocesora, np. wskaźnik stosu, SP.

Podobne przygotowania powinny być przeprowadzone w wybranym obszarze pamięci RAM, który będzie używany przez urządzenia I/O oraz działania typowo programowe. Do pamięci RAM powinno wprowadzić się takie dane, które umożliwią poprawną obsługę każdego zdarzenia, natychmiast po jego pojawieniu się.

Wszystkie wskazane wyżej czynności należą do działań, które nazywa się przygotowaniem stanu środowiska.



przygotowanie środowiska to zespół działań przygotowujących to środowisko (urządzenia I/O i zadania programowe) do pracy w pętli programowej ..

Najczęściej spotykanym sposobem przygotowania środowiska jest takie jego ustawienie, by w momencie wejścia do obszaru pętli wszystkie flagi zdarzeń były ustawione w stan pasywny. Nie jest to jednak regułą. Zdarza się, że niektóre urządzenia mikroprocesorowe sygnalizują podjęcie normalnej pracy (pracy w obszarze pętli) przez wygenerowanie, np. krótkiego sygnału dźwiękowego czy też wyświetlenia wstępnego komunikatu, np. na wskaźniku LCD. Ponieważ fakt włączenia generatora dźwięku lub fakt rozpoczęcia przekazywania danych do wskaźnika LCD są zdarzeniami, ich flagi powinny być aktywowane w ramach przygotowania środowiska.



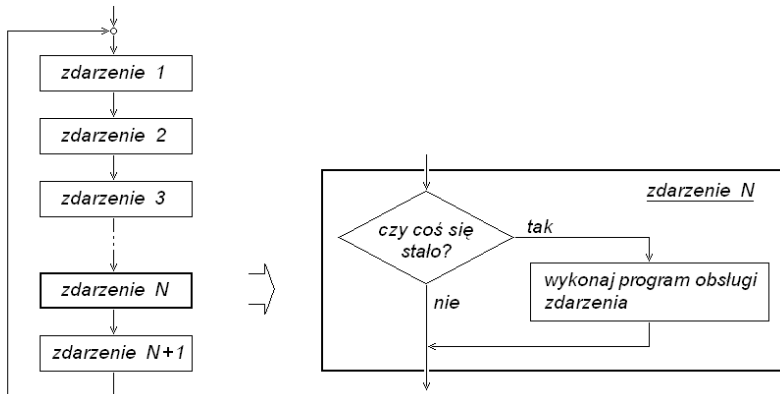
Rys. 1.4.1. Etapy wykonywania programu: przygotowanie środowiska i praca w pętli.

Reasumując, cały program mikrokontrolera da się umiejscowić w dwu blokach funkcjonalnych: w bloku przygotowania środowiska oraz w pętli programowej. Ze względu na wygodę omawiania problemów przypisywanych pętli programowej, blok funkcjonalny pętli programowej, pokazany na rys.1.4.1, może być dalej nazywany **pętlą główną programu**.

1.4.3. Obsługa zdarzeń w pętli programowej.

Testowanie stanu środowiska i obsługa zdarzeń.

W ramach czynności objętych pojęciem pętli, program komputera dokonuje przeglądu stanu środowiska i po zaobserwowaniu faktu wystąpienia zdarzenia obsługuje je. Typowy sposób wykonywania programu w obszarze pętli pokazano na rys.1.4.2. Testowanie stanu środowiska odbywa się w sposób sekwencyjny - stan poszczególnych zdarzeń jest testowany w kolejności, ustalonej przez programistę w momencie pisania programu. Obserwacja musi być prowadzona w ten sposób, by każde zdarzenie było zauważone i właściwie obsłużone - nie ma mowy o przeoczeniu jakiegokolwiek z nich. Ponieważ częstość występowania każdego ze zdarzeń jest limitowana od dołu, do rozwiązania problemu wystarczy odpowiednio częste testowanie stanu środowiska (więcej informacji o tym problemie podano w podrozdziale "Czas obsługi zdarzenia")



Rys. 1.4.2. Etapy wykonywania programu pętli - obserwacja środowiska.

Poszukiwanie zdarzeń zazwyczaj odbywa się poprzez testowanie stanu flagi przypisanej tym zdarzeniom. Sprawdzanie stanu flagi jest procesem bardzo szybkim. Przykładowy fragment programu testowania środowiska, napisany w języku assemblera mikrokontrolera 80C51, może wyglądać następująco:

Program 1.4.1.

```

.      jnb   flaga_1, dalej_2      ; dalej gdy brak zdarzenia 1      (1)
.      clr   flaga_1              ; kasuj flagę zdarzenia 1      (2)
.      lcall  obsluga_zdarzenia_1  ; wykonaj obsługę zdarzenia 1  (3)
dalej_2:
.      jnb   flaga_2, dalej_3      ; dalej gdy brak zdarzenia 2      (4)
.      clr   flaga_2              ; kasuj flagę zdarzenia 2      (5)
.      lcall  obsluga_zdarzenia_2  ; wykonaj obsługę zdarzenia 2  (6)
dalej_3:
.      xxx                          ; dalszy kod pętli programowej  (7)

```

Jak szybko jest przeglądany stan środowiska? Wykonanie instrukcji, np. *jnb flaga_1, dalej_2* jest realizowane w 2 cyklach maszynowych (24 cykle zegarowe). Dla mikrokontrolera 80C51, pracującego z zegarem 11,059MHz, czas wykonania tej instrukcji wynosi 2,17 μ s. Jeżeli w małym systemie mikroprocesorowym może wystąpić 10 zdarzeń, to czas przepytывania całego środowiska wyniesie około 22 μ s - środowisko w takim przypadku będzie testowane ponad 45000 razy na sekundę. Oczywiście, jedynie w przypadku braku zdarzeń (braku obsługi tych zdarzeń).

W podanym wyżej przykładzie, po zauważeniu stanu aktywnego flagi, do wykonywanego kodu testowania stanu środowiska jest dołączany kod obsługi zdarzenia (instr. *lcall*) a stan flagi jest kasowany. Częstość obserwacji środowiska, w takim przypadku, zmniejsza się.

Pętla w pętli.

W każdym procesie programowania znajdzie się przypadek, który zmusza program do oczekiwania na zakończenie innych działań. Przykładem może być zlecenie wysłania grupy bajtów przez port transmisji szeregowej w trybie asynchronicznym. Nadajnik typowego portu transmisji szeregowej jest tak skonstruowany, że z punktu widzenia mikroprocesora stanowi on rejestr do zapisu (SBUF w 80C51, patrz rozdział 1.2.6). Po wprowadzeniu danej do bufora nadajnika, stan tego rejestru nie powinien być zmieniany do momentu wysłania wszystkich bitów danej. Wysłanie następnego bajtu może nastąpić dopiero po zakończeniu wysyłania bajtu poprzedniego. Wypadałoby zatem poczekać na właściwy moment, w którym można by wysłać następny bajt. Można to osiągnąć poprzez utworzenie dodatkowej **pętli w pętli** głównej programu. Pętla dodatkowa w tym wypadku jest pętlą pomocniczą, umożliwiającą wykonanie obsługi zdarzenia.

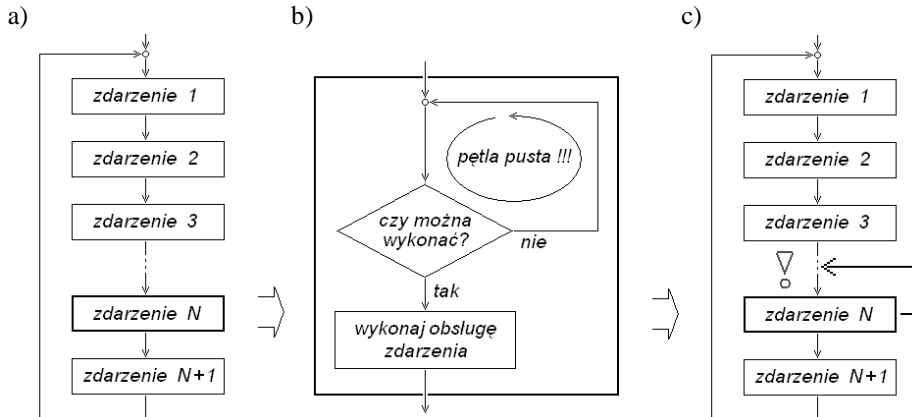


pętla pomocnicza to każda pętla wbudowana w pętlę główną programu ..

Opisywaną sytuację pokazano na rys.1.4.3. Pętla pomocnicza, oprócz pytania o stan nadajnika nie zawiera jakichkolwiek instrukcji - pętla **jest pusta**. Jeżeli sprzęt mikrokomputera działa poprawnie to po pewnym czasie nastąpi wysłanie kolejnego bajtu. Jeżeli jednak sprzęt zawiedzie, a konkretnie nadajnik nie będzie w stanie przekazać informacji o swoim stanie to nastąpi zawieszenie wykonywania programu (!) - w takim przypadku, w nieskończoność, wykonywane będzie pytanie o stan nadajnika. Można zatem stwierdzić, że pętla pusta, bez żadnych zabezpieczeń, może stanowić zagrożenie dla pracy komputera.



pusta pętla pomocnicza w pętli programowej to **błąd** - może przyczynić się do zawieszenia wykonywania programu ..



Rys. 1.4.3. Problem "pętla w pętli" - pętla pusta.

Istnieje też inny problem, zupełnie nie związany z wadami sprzętu. Załóżmy, że szybkość nadawania jest bardzo mała i wynosi np. 75 bodów. Przy dodatkowym założeniu, że wysyłana dana składa się z 8 bitów, jest wysyłany 1 bit stopu i wyłączona jest kontrola parzystości/nieparzystości, wysyłanie bajtu (pakietu 10 bitów) będzie trwało ponad 133 ms. Oznacza to, że środowisko systemu mikroprocesorowego będzie obserwowane raz na 133 ms - zaledwie 7,5 razy na sekundę. Z punktu widzenia systemu mikroprocesorowego, jest to niedopuszczalnie długi czas i duża grupa urządzeń pozostałaby bez obsługi.



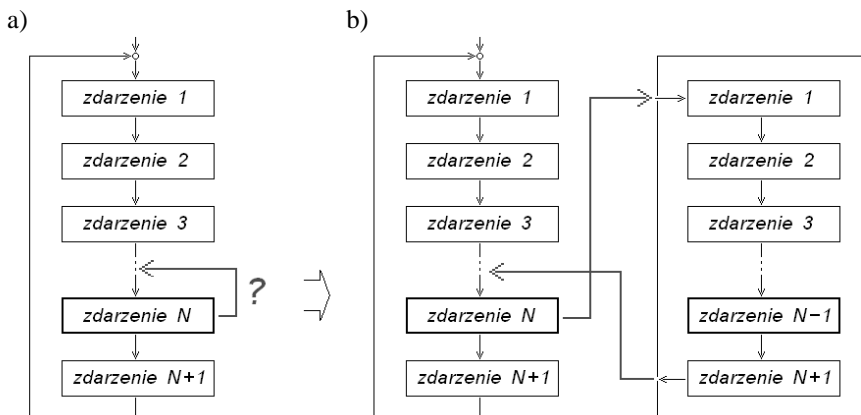
pusta pętla pomocnicza w pętli programowej to **błąd** - może spowodować obniżenie częstości obserwacji środowiska ..

Co zatem robić? Najprostszym rozwiązaniem wydaje się wprowadzenie do pętli pustej "pomocniczego" testowania stanu środowiska. W takim przypadku, do pętli pustej powinno wprowadzić się poszukiwanie wszystkich zdarzeń oprócz tego, dla którego pętla pusta powstała. Po takiej operacji, pętla programowa wglądałaby tak, jak pokazano na rys.1.4.4.

Jak widać z rysunku, problem obserwacji środowiska rzeczywiście będzie rozwiązany i wszystkie zdarzenia będą mogły być dostrzeżone. Problem jednak w tym, że program pętli, za wyjątkiem zdarzenia N, został **zdublowany**. Jakakolwiek zmiana treści programu musiałaby być przeniesiona do innych, skopionych struktur.

A co będzie, gdy podobny mechanizm trzeba będzie zastosować do obsługi innych zdarzeń? Jeżeli tak, to podobną strukturę, jak pokazana na rys.1.4.4b, należałoby wbudować zarówno w główną pętlę programową jak i w pętlę pomocniczą. A to może być naprawdę trudne do wykonania, nawet w przypadku 2 zdarzeń. Bo jeżeli, np., oczekuje się na zakończenie obsługi zdarzenia N i w ramach tego oczekiwania znajdzie potrzeba obsługi innego zdarzenia, to zachodzi pytanie, jak w drugiej pętli pomocniczej uniknąć testowania zdarzenia N?(!).

I na odwrót. Wniosek z powyższych rozważań nasuwa się sam: wbudowanie w strukturę pętli pomocniczej elementów pętli głównej jest błędem.



Rys. 1.4.4. Problem "pętla w pętli" - pętla pomocnicza testowania środowiska.



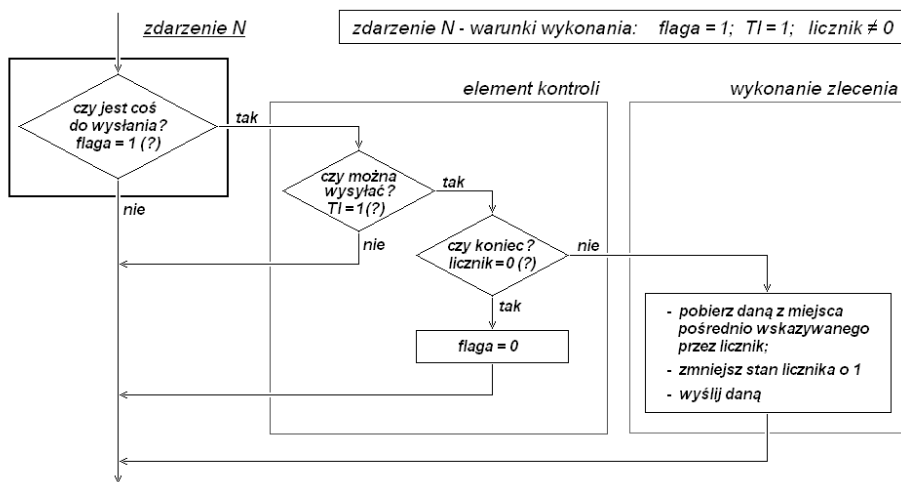
pętla pomocnicza, wbudowana w pętlę główną i zawierająca elementy pętli głównej to **błąd ..**

Problem opisany wyżej da się rozwiązać w bardzo prosty sposób. Wystąpienie zdarzenia N ustawia jego flagę w stan aktywny. Po zaobserwowaniu stanu aktywnego flagi, następuje próba wykonania programu obsługi zdarzenia. Nie wykonanie za wszelką cenę ale próba wykonania. W przypadku braku możliwości natychmiastowego wykonania tej obsługi, jest ona odkładana na czas późniejszy. Wystarczy w tym celu nie zmieniać stanu flagi. Ponowienie próby obsługi zdarzenia, ze względu na niezmienny i dalej aktywny stan flagi, będzie wykonane w następnym cyklu testowania środowiska (kolejne wykonanie zadań pętli). Można zauważyć, że brak możliwości wykonania obsługi zdarzenia skutkuje, w takim przypadku, pełnym testowaniem stanu środowiska. Dalszy brak możliwości obsługi zdarzenia powoduje powtórzenie opisanej sytuacji aż do momentu, w którym obsługa będzie wykonana i flaga zdarzenia zostanie ustawiona w stan pasywny. Podsumowując, opisany sposób postępowania powoduje, że wykonanie programu obsługi zdarzenia jest przesuwane do momentu, w którym to wykonanie będzie możliwe. Dzięki temu środowisko jest testowane w sposób ciągły a do pętli nie wprowadza się pętli pomocniczych, zawierających elementy pętli głównej.

Opisany wyżej sposób rozwiązania problemu najlepiej jest prześledzić na przykładzie. Niech zdarzeniem N będzie zlecenie wysłania, np., kilku bajtów przez port transmisji szeregowej. Bit flagowy tego zdarzenia może nosić nazwę *flaga*. Liczbę bajtów do wysłania można umieścić w bajcie kontrolnym (pomocniczym) zdarzenia o nazwie *licznik*. Bit flagowy nadajnika portu transmisji sze-

regowej może nazywać się *TI* (jak w 80C51). Bit *TI* powinien sygnalizować gotowość do nadajnika do przyjęcia nowej danej - na pewno taki stan osiąga się po zakończeniu wysyłania poprzedniego bajtu. Diagram kolejnych działań podano na rys.1.4.5 a odpowiadający mu fragment kodu w przykładzie programu 1.4.2.

Dla środowiska systemu mikroprocesorowego, zdarzeniem jest grupa bajtów przeznaczonych do wysłania, która powinna być wysłana przez port transmisji szeregowej i jeszcze nie została wysłana. Obecność zdarzenia jest sygnalizowana ustawionym bitem *flaga*. Wszystkie pozostałe czynności, pokazane na rys. 1.4.5, należą do obsługi zdarzenia. Ponieważ proces wysyłania jest zależny od działania sprzętu oraz liczby bajtów do wysłania, program obsługi zdarzenia można podzielić na dwie fazy: fazę zapytań (kontroli) oraz fazę wykonania. Na rys.1.4.5, pokazano obie fazy obsługi zdarzenia - są one objęte ramkami. Pierwszym pytaniem postawionym w fazie kontroli powinno być pytanie o to, czy sprzęt jest gotowy do wysłania bajtu (czy $TI=1$). Brak gotowości sprzętu powinien spowodować natychmiastowe zaniechanie dalszej obsługi zdarzenia. Potwierdzenie gotowości nadajnika do wysłania bajtu powinno spowodować następane pytanie o to, czy jest coś do wysłania (czy licznik = 0). Informacja o braku bajtów do wysłania (licznik=0) powinna spowodować skasowanie stanu aktywnego flagi zdarzenia i natychmiastowe zakończenie obsługi zdarzenia. Jeżeli jednak zapytanie o liczbę bajtów do wysłania da wynik pozytywny (licznik \neq 0), to powinno nastąpić pobranie bajtu z miejsca jego przebywania, wysłanie go oraz zmniejszenie stanu licznika wysłanych bajtów.



Rys. 1.4.5. Wysyłanie bajtów przez port transmisji szeregowej.

Przedstawiony na rys.1.4.5 schemat postępowania, zapisany w języku assemblera, może wyglądać następująco:

Program 1.4.2.

<i>;</i>	<i>testowanie:</i>		
	<i>jnb</i>	<i>flaga, dalej</i>	<i>; dalej gdy brak zdarzenia (1)</i>
	<i>lcall</i>	<i>obsługa_zdarzenia</i>	<i>; wykonaj obsługę zdarzenia (2)</i>
<i>dalej:</i>			
	<i>...</i>		<i>; dalszy kod pętli programowej (3)</i>
<i>;</i>	<i>obsługa_zdarzenia:</i>		
	<i>jb</i>	<i>TI, wyslij_1</i>	<i>; testuj flagę portu nadajnika (4)</i>
	<i>ret</i>		<i>; wróć gdy nie można wysłać (5)</i>
<i>wyslij_1:</i>			
	<i>mov</i>	<i>a,licznik</i>	<i>; odczytaj stan licznika (6)</i>
	<i>jnz</i>	<i>wyslij_2</i>	<i>; testuj stan licznika (7)</i>
	<i>clr</i>	<i>flaga</i>	<i>; kasuj flagę zdarzenia (8)</i>
	<i>ret</i>		<i>; wróć gdy nie ma nic do wysłania (9)</i>
<i>wyslij_2:</i>			
	<i>dec</i>	<i>licznik</i>	<i>; zmniejsz stan licznika (10)</i>
	<i>...</i>		<i>; pobierz bajt i wyślij go *** (11)</i>
	<i>ret</i>		<i>; wróć po wysłaniu bajtu (12)</i>

W linii 1 programu zadawane jest podstawowe pytanie o istnienie zdarzenia. Pozostałe linie, zgrupowane pod wspólną nazwą *obsługa_zdarzenia* są programem obsługi zdarzenia. Linie 4..9 należą do fazy kontrolnej programu obsługi, a linie 10..12, do fazy wykonania. Linia 11 ukrywa kilka dodatkowych instrukcji związanych z ustaleniem adresu bajtu do wysłania, pobrania tego bajtu i faktu jego wysłania.

Jak wpływa przedstawiony wyżej schemat wysyłania bajtów na sam proces wysyłania jako taki. Można łatwo zauważyć, że wykonanie zlecenia wysłania grupy bajtów może trwać dłużej niż wynikałoby to z parametrów transmisyjnych (szybkości transmisji). Jest to związane z dodaniem do czasu wysyłania poszczególnych bajtów średniego czasu przepytывania stanu środowiska i obsługi jego zdarzeń. Zakładając, że szybkość transmisji asynchronicznej wynosi 9600 bodów, że wysyłanych jest 8 bitów danej, że występuje jeden bit stopu oraz wyłączono kontrolę parzystości/nieparzystości, czas wysyłania pojedynczego bajtu wyniesie 1,04 ms. Zakładając ponadto, że średni czas obserwacji środowiska wyniesie, np. 100 μs, czas wysyłania pojedynczego bajtu wzrośnie do wartości 1,14 ms, tj. o około 10% w stosunku do sytuacji idealnej. Dla tego typu transmisji, zwiększenie czasu transmisji o 10% nie ma praktycznie żadnego znaczenia. W przypadkach szczególnych, w których problem zwiększenia czasu transmisji jest elementem bardzo ważnym, można proces transmisji usprawnić poprzez użycie przerwań.

Powracając do problemu "pętla w pętli" można zadać pytanie: czy wobec uwag poczynionych wyżej, jest jednak możliwe wprowadzenie pętli pomocniczej do obsługi zdarzenia? Oczywiście, że tak. Jeżeli program obsługi zdarzenia

zawiera N identycznych działań to warto je wykonać za pośrednictwem pętli pomocniczej. Pętla taka **nie zawiera pytań o stan środowiska** co powoduje, że znikają wszelkie wątpliwości, jak to środowisko obserwować i obsłużyć. O ile czas wykonania pętli pomocniczej nie przekracza dozwolonych wielkości, obecność takiej pętli w obszarze pętli głównej jest jak najbardziej dozwolona a czasami wręcz niezbędna. Jeżeli czas wykonania jest długi, można obsługę zdarzenia wykonać w sposób przedstawiony w następnym punkcie tego rozdziału.



pętla pomocnicza, po spełnieniu określonych warunków, nie musi być elementem groźnym dla środowiska ..

I jeszcze jedna uwaga. Bardzo często, w literaturze (np. [3]), w ramach nauki programowania, wprowadza się, a później często używa, procedury typu DELAY albo WAIT. Procedury te, nazywane procedurami opóźniającymi, są pętlami pomocniczymi, w których dba się jedynie o wykonanie odpowiedniej liczby instrukcji co bezpośrednio wpływa na czas wykonania procedury. Instrukcje tych procedur nie mają nic wspólnego z testowaniem stanu środowiska. Procedury są wywoływane w celu opóźnienia wykonania jakiegoś działania w strukturze środowiska i opóźnienie przez nie wprowadzane może dochodzić nawet do kilkuset milisekund. W czasie wykonywania procedury opóźniającej środowisko nie jest obserwowane - można stracić bardzo ważne informacje. A to jest sprzeczne z ideą sensownej obsługi środowiska. Oczywiście, czasami zachodzi konieczność przeprowadzenia szybkiego testu czasowego i wykorzystanie prostej procedury opóźniającej nie jest czymś nagannym. W przypadku jednak konieczności stosowania kontrolowanych opóźnień w działaniu środowiska, należy do tego celu stosować tak skonstruowane procedury opóźniające by nie stanowiły one problemu dla środowiska ale były kolejnym zdarzeniem tego środowiska (patrz zadanie laboratoryjne 1).



procedura opóźniająca, będąca pętlą pomocniczą, w której brakuje obserwacji środowiska, to **błąd** ..

Obsługa zdarzeń powiązanych.

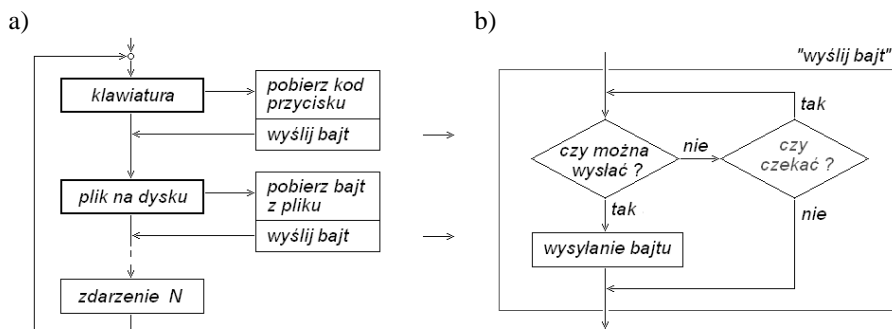
Następnym z problemów procesu programowania systemów mikroprocesorowych jest problem zdarzeń wzajemnie powiązanych. Jak już wspomniano, każde zdarzenie powinno posiadać własny program obsługi. Słowo *własny* podkreśla, w tym przypadku, procedurę najbardziej optymalną dla konkretnego zdarzenia. Bardzo często jednak, niektóre zdarzenia są ściśle powiązane z innymi zdarzeniami środowiska. Problem ten i sposób jego rozwiązania, najwygodniej będzie prześledzić za pomocą przykładu.

Założmy, że ogólnie pojętym zdarzeniem będzie konieczność wysyłania bajtów przez port transmisji szeregowej. Założmy ponadto, że istnieją dwa

miejsca pojawiania się tych bajtów. Jednym z nich będzie klawiatura systemu mikroprocesorowego a drugim zawartość pliku, umieszczonego np. na dysku systemu. Można w takim przypadku mówić o 3 różnych zdarzeniach. Pierwszym z nich będzie pozyskiwanie bajtów z klawiatury. Drugim zdarzeniem będzie odczytywanie stanu pliku. Trzecim wreszcie zdarzeniem będzie wysyłanie bajtów przez port transmisji szeregowej. Można zauważyć, że zdarzenia 1 i 2, w wyniku ich obsługi, spowodują pojawienie się bajtów do wysłania, które to wysłanie można wykonać dzięki obsłudze zdarzenia 3. Można również zauważyć i to, że w obu przypadkach będzie niezbędne powiązanie zdarzeń - do zdarzeń 1 i 2 musi być dodane zdarzenie nr 3. Opisowaną sytuację przedstawiono na rys.1.4.6. Na pierwszy rzut oka, nie widać żadnych problemów z tym związanymi - trzeba przecież pobrać bajt i wysłać go. Problemy jednak istnieją.

Załóżmy, że transmisja szeregową jest opisana parametrami: 8 bitów danych, 1 bit stopu, wyłączona jest kontrola parzystości/nieparzystości a szybkość transmisji to 75 bodów. W takim przypadku, wysyłanie kolejnych bajtów może odbywać się nie częściej niż co 133 ms - maksymalnie 7,5 razy na sekundę.

Klawiatura nie należy do "szybkich" urządzeń I/O. Ze względu na to, że nie jesteśmy w stanie zbyt często naciskać na jej przyciski, program obsługi klawiatury nie musi być wykonywany zbyt często. By dać jednak szansę najszybciej piszącemu, ustalenie maksymalnej liczby naciśnień, np. na wielkość 20 razy na sekundę, wydaje się być rozsądnym założeniem. Dzięki temu, system mikroprocesorowy rozpozna 20 naciśnień i 20 zwolnień nacisku na przycisk klawiatury. System może wygenerować 20 kodów na sekundę - co 50 ms może pojawić się informacja o nowym stanie klawiatury. Porównując ten czas z czasem wysyłania bajtu, można zauważyć, że w trakcie wysyłania jednego bajtu przez port szeregowy, klawiatura może "wytworzyć" ponad 2 bajty (2,6 bajta).



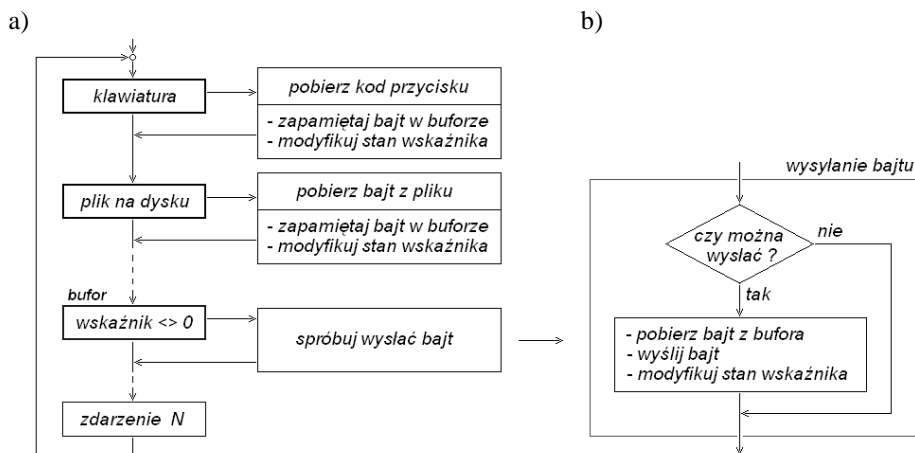
Rys. 1.4.6. Problem zdarzeń powiązanych - problem decyzji.

Przyglądając się obsłudze zdarzenia, opisanej na rys.1.4.7 nazwą *klawiatura*, można zauważyć, że problem obsługi jest umiejscowiony w elemencie *wyslij bajt*. Występuje on w przypadku, gdy sprzęt nie jest gotowy do wysłania bajtu bo, np. jest wysyłany poprzedni bajt. Nie jest to problem środowiska - jest to problem programisty. Musi on zdecydować, jak ma wyglądać ta część

programu: czy należy czekać na gotowość portu do wysłania danej czy też zaniechać wysłania pozyskanego z klawiatury bajtu. Czekanie na gotowość portu wiąże się ze stworzeniem pętli pustej, co zostało zdecydowanie skrytykowane wcześniej (patrz podrozdział "Pętla w pętli"). Bo co się stanie, gdy w czasie oczekiwania na wysłanie bajtu naciśniemy na przycisk klawiatury jeszcze raz? To zdarzenie nie zostanie zauważone - stracimy bajt. Zrezygnowanie z oczekiwania też nie wchodzi w rachubę - nie wolno utracić żadnej danej a dana jest i czeka na wysłanie.

Identyczny problem występuje przy obsłudze zdarzenia, opisanego na rys. 1.4.6a nazwą *plik na dysku*. Dane z pliku są pobierane bardzo szybko w porównaniu danymi pozyskiwanymi z klawiatury i proces ewentualnego "hamowania" przez pętlę pustą jest, w takim wypadku, szczególnie jaskrawy.

Jedynym sensownym sposobem rozwiązania powyżej opisywanego problemu jest rozdzielanie zdarzeń powiązanych. Rozwiązanie, w sposób schematyczny, pokazano na rys.1.4.7. Rozdział zdarzeń może być dokonany poprzez wprowadzenie elementu pośredniczącego, nazywanego dalej *buforem*. Bufor to zarezerwowane pole pamięci RAM, w którym będą składowane dane do wysłania. Obsługa zdarzeń *klawiatura* i *plik na dysku*, w takim wypadku, może być sprowadzona do pobrania danej i zapisania jej w pamięci RAM - w buforze. Mechanizm obsługi zdarzeń jest identyczny i sprowadza się do trzech czynności: do pobrania bajtu, do zapisania bajtu w buforze pod adres *stan wskaźnika* oraz do modyfikacji tego adresu. Można zauważyć, że dzięki takiemu postępowaniu, omawiane zdarzenia pozbyły się elementu wysłania bajtu i problemów z tym związanych - wysłanie zostało zastąpione zapisywaniem danej do pamięci.



Rys. 1.4.7. Rozwiązanie problemu zdarzeń powiązanych.

Bajty zgromadzone w buforze muszą być wysłane. Fakt istnienia takich bajtów powinien spowodować zapytanie o możliwość ich wysłania. Uzyskanie zgody powinno spowodować pobranie danej z bufora i jej wysłanie. Brak zgody powinien skutkować zaniechaniem wszelkich działań. W takim wypadku, dana

nie zostanie zagubiona i można podjąć próbę jej wysłania w innym czasie. Z punktu widzenia środowiska, takie działanie jest zdarzeniem i powinno być obsługiwane w obszarze pętli programowej. W przedstawionym na rys.1.4.7a schemacie pętli, oznaczono je nazwą *bufor*. Sposób obsługi zdarzenia pokazano na rys.1.4.7b.

Nowo wprowadzone do pętli zdarzenie *bufor* nie jest niczym nowym dla środowiska. Występowało ono w powiązaniu ze zdarzeniami *klawiatura* i *plik na dysku*. Wprowadzenie bufora danych jako elementu pośredniczącego pozwoliło jedynie na separację zdarzeń powiązanych - pozwoliło na uporządkowanie działań w środowisku.



rozdzielanie zdarzeń powiązanych to mechanizm porządkowania środowiska ..

Czas obsługi zdarzenia.

Program mikrokomputera jest wykonywany w obszarze pętli programowej. Obszar pętli powinien być bezustannie testowany w celu znalezienia zdarzeń wymagających obsługi. O ile proces samego testowania może być sprowadzony do wykonania pojedynczej instrukcji (testowanie stanu flagi) o tyle sama obsługa zdarzenia może okazać się bardzo skomplikowana i wymagać zastosowania setek albo nawet tysięcy instrukcji. Ponieważ czas wykonania obsługi jest sumą czasów wykonywania poszczególnych instrukcji, czas obsługi zdarzenia może okazać się bardzo długi. Rozpoczęcie i wykonywanie obsługi takiego zdarzenia może spowodować blokadę testowania stanu środowiska na długi czas - a to jest poważny błąd.

Użyte wyżej słowo *długi* jest związane z czasem wykonania programu i jest pojęciem względnym - w każdym przypadku jego użycia powinno się podać relację do wzorca czasu. Jeżeli jakieś zdarzenie należy do zdarzeń, które pojawia się w środowisku najczęściej, to czasowym *puntem odniesienia* może być najkrótszy z czasów pomiędzy kolejnymi powtórzeniami tego zdarzenia. Taki czas można nazwać czasem krytycznym - środowisko na pewno zostanie obsłużone gdy maksymalny czas obsługi wszystkich zdarzeń, które pojawiły się jednocześnie, będzie krótszy od czasu krytycznego.



czas krytyczny to minimalny odstęp czasu pomiędzy kolejnymi wystąpieniami zdarzenia, które w środowisku pojawia się najczęściej ..

Dla uproszczenia dalszego opisu, w dalszej części skryptu będzie używane pojęcie programu krótkiego oraz programu długiego. Program krótki to taki, którego czas wykonywania jest zdecydowanie krótszy od czasu krytycznego. Program długi to program, którego czas wykonywania jest porównywalny lub

dłuższy od czasu krytycznego. Oczywiście, pojęcie programu krótkiego jest powiązane z innymi programami obsługi. To suma czasu wykonania tych programów, jeden po drugim, odniesiona do pojęcia czasu krytycznego może stanowić o "krótkości" każdego z nich. Określenia: program długi i program krótki, można przenieść na pojęcie obsługi zdarzenia - można mówić, w takim przypadku, o krótkiej lub długiej obsłudze zdarzenia.



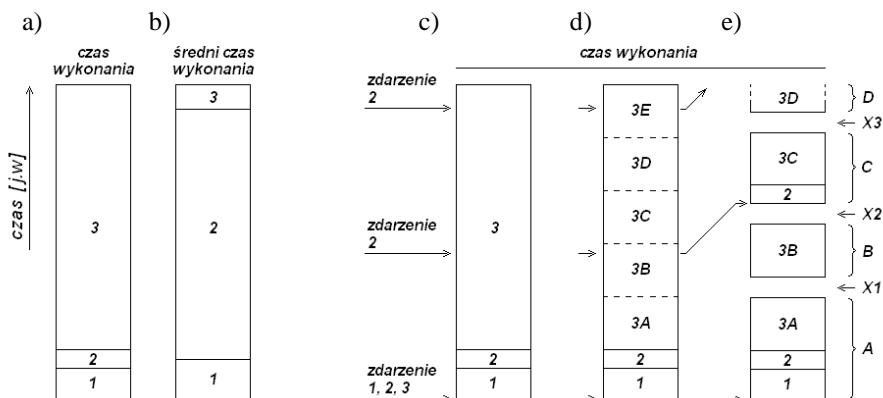
program długi to program obsługi zdarzenia, którego wykonanie zajmuje na tyle długi czas, że może to spowodować kłopoty w procesie obsługi środowiska ..

program krótki to program obsługi zdarzenia, który nie stwarza kłopotów w procesie obsługi środowiska ..

A co się stanie, gdy pojedynczy program obsługi zdarzenia (lub grupa programów) nie spełni warunku programu krótkiego? Przykładem takiego zdarzenia może być odczyt grupy bajtów z zewnętrznej, szeregowej pamięci EEPROM z łączem I²C (patrz zadanie laboratoryjne 6). Zakładając, że maksymalna częstota odczytu wynosi 100 kb/s, odczytanie 1 bajtu zajmie ok. 90μs. Odczytanie, np. 100 bajtów to czas 9ms. Czas krytyczny systemu może być wyznaczony np. przez zdarzenie odbioru bajtów, przesyłanych portem transmisji szeregowej, który pracuje w trybie asynchronicznym. Jeżeli przesyłane jest 8 bitów danej, 1 bit stopu, wyłączona jest kontrola parzystości/nieparzystości a szybkość transmisji wynosi 19200b/s to czas krytyczny zdarzenia wynosi ok. 500μs. Porównując czas krytyczny z czasem odbioru pojedynczego bajtu łączem I²C widać, że odbiór bajtu może należeć do grupy programów krótkich ale odbiór 100 bajtów już nie.

Opisany wyżej przykład może sugerować, że przesłanie 100 bajtów magistralą I²C może być, wobec wcześniej uczynionych stwierdzeń, praktycznie niemożliwe. Niekoniecznie. Do rozwiązania problemu wystarczy odpowiednia organizacja struktury programu. Sposób rozwiązania problemu przedstawiono na rysunku 1.4.8.

Poszczególne zdarzenia przedstawione są na rysunku w postaci numerowanych bloków. Czas wykonania programu obsługi poszczególnych zdarzeń jest reprezentowany przez wysokość bloków (czas w jednostkach względnych). Przy założeniu, że wystąpiła potrzeba obsługi grupy zdarzeń, czas obsługi tej grupy jest sumą czasów przypisanych elementom grupy (rys.1.4.8a). Jest to czas rzeczywisty wykonania obsługi grupy zdarzeń. Przyglądając się tym samym zdarzeniom w dłuższym przedziale czasu, można zbudować wykres, pokazany na rys.1.4.8b. Wysokość poszczególnych bloków uzyskano przez sumowanie czasów obsługiwanych w tym czasie zdarzeń. Okazuje się, że obraz rozkładu czasu obsługi zdarzeń jest kompletnie różny od tego, pokazanego na rys.1.4.8a. Nie ma w tym nic dziwnego ponieważ poszczególne zdarzenia występują z różną częstością.



Rys. 1.4.8. Problem obsługi zdarzenia "długiego".

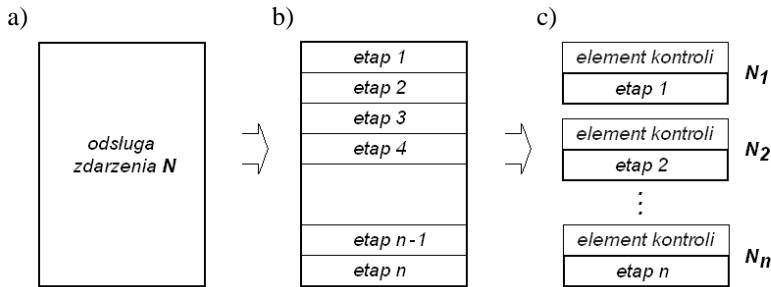
To, że niektóre zdarzenia występują stosunkowo rzadko, pozwala na proste rozwiązanie problemu obsługi zdarzeń opisanych programem długim. Na rys. 1.4.8c pokazano miejsca występowania zdarzeń. Jeżeli w umownej chwili T_0 pojawią się zdarzenia 1, 2 i 3, to powinny być one kolejno obsługiwane, w kolejności np. 1, 2 i 3. Jeżeli program obsługi zdarzenia 3 jest programem długim, to pojawiające się w czasie jego wykonywania, dwukrotne żądanie obsługi zdarzenia 2 może być niezauważone - wystąpi błąd obsługi środowiska. Z rys.1.4.8b wynika jednak, że obsługa zdarzenia 3 występuje bardzo rzadko. W takim przypadku, program obsługi zdarzenia 3 można podzielić na kilka elementów (rys.1.4.8d) i wykonywać go tak, jak pokazano na rysunku 1.4.8e. W odróżnieniu od schematu postępowania z rys.1.4.8c, po zaobserwowaniu zdarzeń 1, 2 i 3, wykonywana jest obsługa zdarzeń 1, 2 oraz etapu 3A zdarzenia 3 (miejsce A). Po wykonaniu tej obsługi, w miejscu X1, jest testowany stan środowiska (wykonanie wszystkich testów pętli programowej) . W wyniku testowania powinno być zauważone to, że obsługa zdarzenia 3 nie jest dokończona i powinno się rozpocząć wykonywanie fragmentu 3B obsługi zdarzenia 3 (miejsce B). Jeżeli w czasie wykonywania obsługi pojawi się nowe zdarzenie, to obsługa tego zdarzenia rozpocznie się dopiero po jego zaobserwowaniu, w miejscu ponownego testowania stanu środowiska, X2. Oczywiście, testowanie środowiska powinno rozpoznać fakt, że obsługa zdarzenia 3 nie jest dokończona i powinno rozpocząć się również wykonywanie fragmentu 3C, itd.



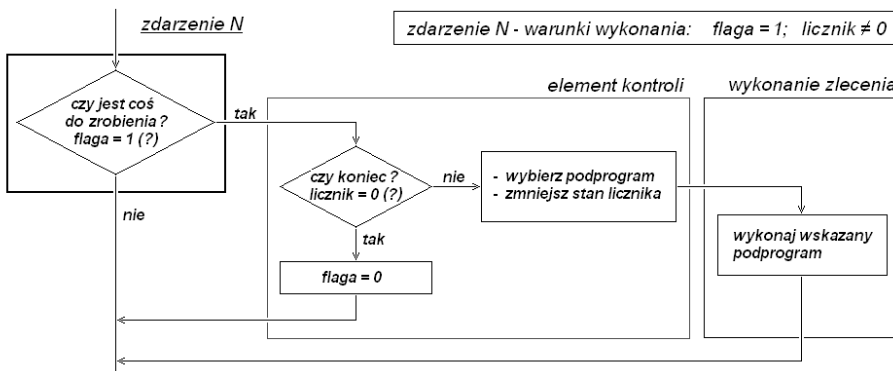
podział programów długich na fragmenty jest sposobem na poprawną obsługę środowiska ..

Przedstawiona wyżej organizacja obsługi zdarzeń pozwala na bezproblemową obsługę tych zdarzeń, które muszą być obsługiwane przez programy długie. Oczywiście, kolejne testowania stanu środowiska, X_n , powinny w takim przypadku rozpoznać fakt braku końca obsługi zdarzenia oraz to, który element podzielonego programu powinien być wykonany.

By zapewnić poprawne administrowanie fragmentami programu, do każdego z nich musi być dodany element kontrolny. Takim elementem może być, np. bajt licznika, wskazujący na wybrany fragment kodu. Sytuację tę przedstawiono na rys.1.4.9. Po dodaniu elementu kontrolnego, obsługa zdarzenia będzie rozpoznawana w środowisku jako szereg zdarzeń N_n .



Rys. 1.4.9. Podział programu długiego na etapy.



Rys. 1.4.10. Obsługa etapów zdarzenia długiego.

Na rys.1.4.10, pokazano, jak może wyglądać obsługa zdarzenia podzielonego. Elementem kontrolnym dla tego przykładu jest bajt licznika, do którego, przed zadeklarowaniem istnienia zdarzenia ($flaga=1$), wprowadzono informację o liczbie fragmentów podzielonego programu. Po wywołaniu programu obsługi zdarzenia N, stan licznika jest wykorzystywany do wskazania fragmentu programu, który ma być wykonany. Po wykonaniu wskazanego kodu, następuje zmniejszenie stanu licznika. Stan licznika równy 0 powoduje zakończenie obsługi zdarzenia N ($flaga=0$). Można zauważyć, że w przedstawionym przykładzie flagą zdarzenia może być sam licznik. W takim przypadku, zapytanie o stan flagi (czy $flaga=1$?) można zastąpić pytaniem o stan licznika: czy $licznik=0$?. Odpowiedź twierdząca, w takim przypadku, będzie oznaczać brak zdarzenia N.

Przedstawiony na rys.1.4.10 schemat postępowania, dedykowany obsłudze zdarzenia podzielonego na 2 fragmenty i zapisany w języku asemblera, może wyglądać następująco:

Program 1.4.3.

```

;   testowanie:
jnb   flaga, dalej           ; dalej gdy brak zdarzenia           (1)
lcall  obsluga_zdarzenia     ; wykonaj obsługę zdarzenia   (2)
dalej:
...                                     ; dalszy kod pętli programowej (3)
;   obsługa_zdarzenia:
mov    a,licznik             ; odczytaj stan licznika      (4)
cjne   a,#2,obsl_1          ; testuj nr zlecenia          (5)
lcall  obsluga_2             ; wykonaj zlecenie           (6)
dec    licznik               ; zmniejsz stan licznika     (7)
ret                                         ; wróć do pętli              (8)
obsl_1:
cjne   a,#1,obsl_0          ; testuj nr zlecenia          (9)
lcall  obsluga_1             ; wykonaj zlecenie           (10)
dec    licznik               ; zmniejsz stan licznika     (11)
ret                                         ; wróć do pętli              (12)
obsl_0:
clr    flaga                 ; kasuj flagę zdarzenia      (13)
ret                                         ; wróć do pętli - koniec zdarzenia (14)

```

W linii 1 programu zadawane jest podstawowe pytanie o istnienie zdarzenia. Pozostałe linie, zgrupowane pod wspólną nazwą *obsługa_zdarzenia* (linia 2) należą do obsługi zdarzenia. Program obsługi zdarzenia jest opisany liniami 4..14. Linie 5, 9 i 13 należą do fazy kontrolnej a linie 6, 7, 10 i 11 do fazy wykonania. Podprogramy *obsługa_1* i *obsługa_2* są fragmentami programu obsługi zdarzenia N.

Można zauważyć, że w przypadku gdy *n* fragmentów programu obsługi zdarzenia jest identycznych ze sobą, wykonanie programu obsługi sprowadza się do *n*-krotnego wykonania pojedynczego fragmentu programu (patrz zadania laboratoryjne 3..6).

Obsługa zdarzeń przez przerwanie.

Przerwanie to "sprzętowy" sposób reagowania na zdarzenia. Jest on związany wyłącznie z urządzeniami typu I/O, które są do tego fizycznie przygotowane. Informacja o wystąpieniu zdarzenia jest przesyłana do mikroprocesora w sposób elektryczny za pośrednictwem linii sygnałowej. Sygnał przerwania jest wprowadzany do mikroprocesora wejściem, które najczęściej jest oznaczane nazwą INT (ang. interrupt). W mikrokontrolerze 80C51, takimi liniami są linie INT0 i INT1.

Dotychczas opisywana metoda obsługi zdarzeń opiera się na niekończącym się procesie sukcesywnego poszukiwania zdarzeń, realizowanym przez badanie stanu ich flag - przez przepytywanie środowiska (ang. polling). Wykrycie zdarzenia powodowało wykonanie programu obsługi tego zdarzenia. Ponieważ każdy program obsługi jest powiązany z czasem jego wykonania, testowanie

stanu konkretnego zdarzenia może następować w sposób "nierównomierny". Kolejne testowania stanu N-tego zdarzenia będą powtarzane bardzo szybko w przypadku, gdy brak będzie aktywnych zdarzeń środowiska. W przypadku aktywacji wielu zdarzeń, kolejne testowania stanu N-tego zdarzenia będą się odbywać z dużymi odstępami czasu.

Istotą przerywania jest to, że umożliwia ono obsługę zdarzenia w sposób inny niż pokazywany dotychczas. Przypadek pojawienia się przerywania powoduje natychmiastowe zatrzymanie bieżąco wykonywanego programu, wykonanie tzw. programu obsługi przerywania i ponowny powrót do wykonywania zatrzymanego programu. Ponieważ pojawienie się przerywania jest ściśle powiązane z wystąpieniem zdarzenia, obsługa tego zdarzenia może być wykonana w dowolnym momencie i to natychmiast po jego zaistnieniu. Jest to niezwykle ważne dla niektórych urządzeń, gdzie natychmiast po zaistnieniu zdarzenia powinna nastąpić jego obsługa (patrz, zadane laboratoryjne 1 i 5). Wydaje się zatem, że przerywania powinny być świetnym sposobem na kontrolę stanu środowiska. To prawda, ale z kilkoma zastrzeżeniami.

Problem przekazywania informacji o zdarzeniach za pośrednictwem przerywań jest bardziej skomplikowany niż wydawałoby się to na pierwszy rzut oka. W przypadku wielu źródeł przerywań, system mikroprocesorowy musi być wyposażony w tzw. kontroler przerywań (patrz rozdział 1.2.7). Obsługa zdarzeń przez przerywania narzuca konieczność zadeklarowania, które ze zdarzeń jest zdarzeniem najważniejszym, które zdarzenia są mniej ważne i jaki jest stopień ich ważności - musi nastąpić deklaracja priorytetu zdarzeń. Zdarzenie o priorytecie wyższym może spowodować przerywanie wykonywania obsługi zdarzenia o priorytecie niższym. Zdarzenie o priorytecie niższym nie może przerwać wykonywania obsługi zdarzenia ważniejszego - obsługa zdarzenia o wyższym priorytecie musi być zakończona.



obsługa zdarzeń przez przerywania narzuca obowiązek zadeklarowania priorytetu zdarzenia ..

Z punktu widzenia mikroprocesora, nieważną sprawą jest to, czy wykonywany jest program "normalny" czy też jest to program obsługi przerywania - mikroprocesor musi wykonywać rozkazy. A skoro tak, to pozostają w mocy wszystkie dotychczas poczynione zastrzeżenia związane z faktem wykonywania programu jako takiego. Program obsługi przerywania musi je akceptować.



program obsługi przerywania jest tylko programem i jest on wykonywany tak jak każdy inny - odmienny jest tylko mechanizm jego uruchamiania ..

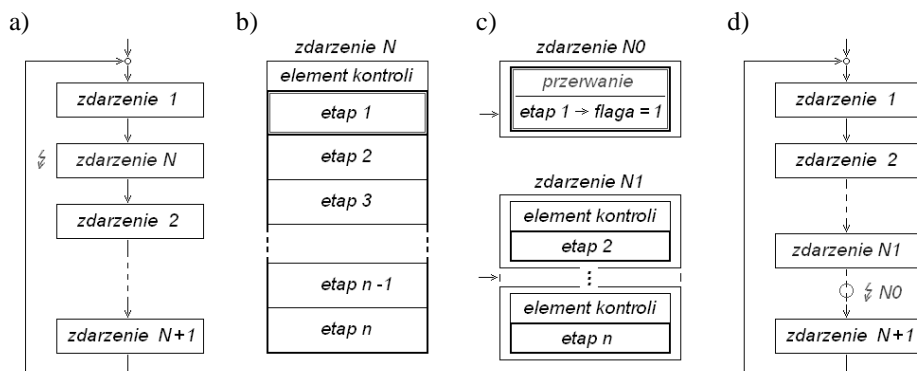
Omawiane do tej pory środowisko składało się zarówno ze zdarzeń powiązanych ze sprzętem jak i zdarzeń typowo programowych, np. obliczeń. Nie ma

potrzeby, by przerwania były generowane przez każde urządzenia I/O (np. klawiatura). Jeżeli urządzenia generujące przerwania istnieją, to środowisko komputera może być obsługiwane zarówno metodą ciągłego przepytывania jak i metodą przerw. Wygenerowanie przerwania powoduje, że w czasie jego wykonywania zablokowana jest możliwość testowania środowiska - a może być błędem. Ponieważ program obsługi przerwania jest dla mikroprocesora takim samym programem jak każdy inny, jego konstrukcja powinna być "przyjazna" dla środowiska. W przypadku procedur przerwaniowych, poprawność konstrukcyjną osiąga się przez minimalizowanie czasu wykonywania tej procedury.



program obsługi przerwania powinien wykonywać czynności wymagające natychmiastowej i bezstratnej obsługi zdarzenia ..

Program obsługi przerwania powinien należeć do grupy programów krótkich (krótki czas wykonania). W przypadku, gdy obsługa urządzenia zgłaszającego przerwanie nie jest prosta i program należy do programów długich, to można postąpić podobnie jak to uczyniono w poprzednim rozdziale - można podzielić program na etapy. Omawianą sytuację przedstawiono na rys.1.4.11.



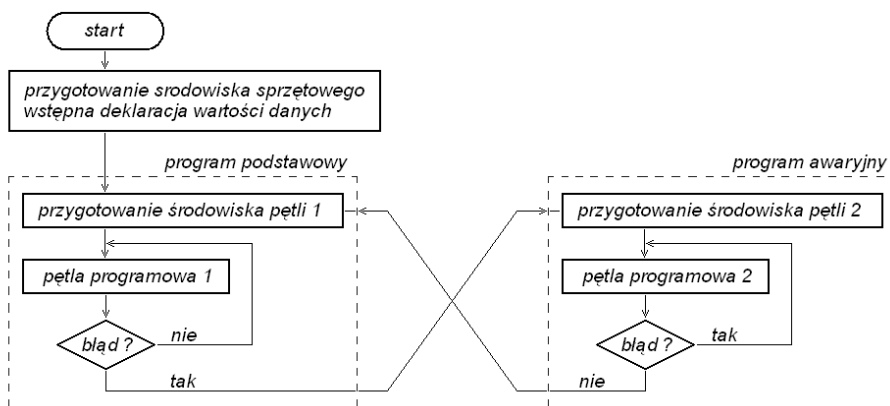
Rys. 1.4.11. Obsługa zdarzenia przez przerwanie.

Etap pierwszy powinien zawierać tylko takie instrukcje, które muszą być wykonane w trybie natychmiastowym, w ramach obsługi przerwania. Ten element obsługi zdarzenia N nazwano zdarzeniem $N0$ (rys.1.4.11c). Pozostałe elementy obsługi zdarzenia zgrupowano w bloku, który nazwano zdarzeniem $N1$. Zdarzenie $N1$ powinno być dostrzeżone i obsłużone w pętli programowej. By stało się to możliwe, w ramach obsługi zdarzenia $N0$, powinno się ustawić flagę aktywacji zdarzenia $N1$ ($flaga=1$). Po dokonaniu tej czynności, pętla programowa będzie wyglądała tak, jak pokazana na rys.1.4.11d. Zdarzenie N jest wykrywane i wykonywane w momencie wymuszonym przez przerwanie ($N0$) a jego kontynuacja ($N1$), w momencie wykrycia jej mechanizmem kolejnego testowania zdarzeń w pętli.

Praca z wieloma pętlami.

Z przeprowadzonych wyżej rozważań wynika, że struktura programu obejmuje dwa elementy: są nimi przygotowanie środowiska i pętla programowa. Wszelkie zadania obsługi zdarzeń są realizowane w obszarze pętli programowej.

Czy jest to jedyny sposób organizacji struktury programu? I tak i nie. Zdarza się, że oprogramowanie mikrokontrolera powinno pracować z różnymi opcjami, np. w trybie pracy "normalnej" oraz trybie pracy "awaryjnej". Często pociąga to za sobą konieczność wykonywania innych zadań, powiązanych z obserwacją innych elementów środowiska i z inną niż w normalnych warunkach intensywnością. Narzuca się logiczny wniosek, że wygodnym sposobem obsługi wskazanych wyżej trybów pracy byłoby stworzenie dwóch ośrodków obsługi zdarzeń - dwu pętli programowych. Każda z pętli, w takim przypadku, będzie przeznaczona do obsługi określonego trybu pracy. Opisywaną sytuację przedstawiono na rys.1.4.12. Pracę w trybie standardowym opisano na rysunku nazwą *program podstawowy* a pracę w trybie awaryjnym, nazwą *program awaryjny*. Węzły zapętlenia o status błędu należą do każdej z dwu pętli.



Rys. 1.4.12. Wykonywania programu w systemie wielopętlowym. - pętle wielokrotne.

Pomimo widocznej na pierwszy rzut oka różnicy pomiędzy strukturą programu pokazaną na rys.1.4.12 oraz 1.4.1, w rzeczywistości nie ma pomiędzy nimi żadnej różnicy. W każdym przypadku, wykonywanie programu odbywa się wyłącznie w obszarze jednej pętli - albo jest wykonywany program podstawowy albo program awaryjny. Nie istnieje w takim przypadku pojęcie "pętla w pętli". Struktura programu z rys.1.4.1 jest zachowana.

Podsumowanie.

Podsumowując przeprowadzone wyżej rozważania, można stwierdzić, że w celu zapewnienia poprawnej obsługi środowiska, program minikomputera powinien być budowany w oparciu o kilka reguł:

- obsługa środowiska powinna być realizowana w pętli programowej - kreowanie w jej obszarze pętli pomocniczych, dublujących zadania pętli programowej, jest błędem;
- stosowanie w obszarze pętli jakichkolwiek procedur opóźniających, utrudniających możliwość obserwacji środowiska, jest błędem;
- jeżeli pewne zdarzenia są pomiędzy sobą powiązane to trzeba je rozdzielić przez wprowadzenie do środowiska zdarzenia pośredniczącego;
- jeżeli długi czas wykonywania programu obsługi zdarzenia może zagrozić obsłudze całości środowiska, to należy ten program podzielić na fragmenty i wykonywać je w kolejnych cyklach testowania stanu środowiska;
- jeżeli obsługa wybranych zdarzeń musi być dokonana w sposób natychmiastowy, to trzeba do tego celu użyć przerw - program obsługi przerwania powinien być jak najkrótszy i prowadzić do wykonania czynności wyłącznie niezbędnych; inne, mniej ważne działania, mogą być wykonane w pętli;

1.2. Programowanie mikrokomputerów.

Programowanie minikomputerów w laboratorium będzie realizowane za pośrednictwem oprogramowania pomocniczego - przy pomocy asemblera i języka C. Język asemblera jest językiem niskiego poziomu - każda linia programu określa pojedynczą instrukcję mikrokontrolera. Zapewnia to najwyższy z możliwych stopień kontroli pracy minikomputera. Wadą programowania w języku asemblera jest to, że treść programu nie może być przeniesiona bezpośrednio na inny typ mikrokontrolera (np. z rodziny MCS-51 na AVR). W przypadku języka wysokiego poziomu, np. C, problem przenoszenia programu na inny typ mikrokontrolera jest stosunkowo łatwy. Wadą tego języka jest to, że w wyniku kompilacji programu, niekoniecznie uzyskuje się kod optymalny dla konkretnego systemu mikroprocesorowego. Inną wadą jest to, że programista, skuszony prostotą kreowania kodu, zapomina o optymalnym sposobie obsługi zdarzeń (patrz poprzedni rozdział), czego efektem może być wadliwa praca minikomputera.

1.2.1. Programowanie w języku asemblera.

Kod źródłowy programu, napisanego w języku asemblera mikrokontrolera, powinien być redagowany przy pomocy dowolnego edytora tekstu, w formacie ASCII. Format zapisu kodu źródłowego jest determinowany programem kompilatora, który będzie użyty do przetłumaczenia tego kodu na kod wynikowy. W laboratorium będzie używany asembler (kompilator) ASEM.W.EXE z pakietu ASEM-51. Plik tekstowy z kodem źródłowym, który będzie tłumaczony przez ten asembler, powinien mieć rozszerzenie A51 (np. TEST.A51). Podane niżej uwagi są związane z asemblerem ASEM-51. Opis ten, z założenia fragmenta-

ryczny, powinien pozwolić na rozpoczęcie pracy z assemblerem - szczegółowe informacje o assemblerze podane są w dokumentacji pakietu ASEM-51 [4].

Ogólne zasady redagowania kodu źródłowego programu.

Poszczególne linie kodu źródłowego zawierają opis wyłącznie jednej instrukcji mikrokontrolera. Dobrą i ogólnie przyjętą zasadą programowania w języku assemblera jest to, że w całym programie jest zachowany ten sam format zapisu linii programu. Elementami formatu są: etykieta (nagłówek linii), mnemonik instrukcji (symboliczny zapis instrukcji), argumenty oraz komentarz. Do pojęcia formatu należy również sposób pisania tekstu - ważnym elementem jest zachowanie, np. takich samych odstępów pomiędzy elementami tekstowymi linii. Odstępy (spacje, tabulatory) nie są brane pod uwagę w procesie kompilacji ale zdecydowanie poprawiają czytelność programu. Typowy wygląd linii programu może być następujący:

```
[etykieta]      [instrukcja][argumenty]   [komentarz]
obsługa_12:    mov    a,#0A2h           ; pobierz wartość stałą (162)
```

Etykieta jest słowem przypisanym instrukcji za nią występującej - jest symbolicznym opisem tej instrukcji. Etykieta wyróżnia instrukcję spośród innych. Etykieta powinna się znajdować po lewej stronie linii programu i być zakończona dwukropkiem. Musi się ona rozpoczynać znakiem litery. Etykiecie, w procesie kompilacji, jest przypisywana wartość liczbowa, która określa adres instrukcji w pamięci programu. Dzięki temu, wszystkie inne instrukcje, które chcą odwołać się do adresu wskazywanego przez etykietę, mogą to uczynić przez podawanie jej nazwy. Etykieta może być wskaźnikiem nie tylko jednej instrukcji - może również wskazywać początek grupy instrukcji, przeznaczonych do wykonania określonego zadania, np. podprogramu.

Następnym elementem linii programu jest instrukcja mikrokontrolera, zapisana w formie symbolicznej. Składnia tekstu instrukcji jest ściśle powiązana z typem mikrokontrolera dla którego jest pisany program. Dane potrzebne do wykonania instrukcji (argumenty instrukcji), mogą być podawane w sposób bezpośredni lub przez nazwy symboliczne, np. etykiety lub przypisania. Dane bezpośrednio muszą być dostosowane do wykonywanej instrukcji i mieć odpowiedni format. Przykładowo, nie można do instrukcji operacji bitowej przydzielać bajtu - daną musi być bit. Jeżeli argumentem instrukcji jest np. bajt, to musi być on poprawnie zapisany - dane liczbowe, dołączane do instrukcji w sposób bezpośredni muszą się rozpoczynać znakiem cyfry. Np. wartość dziesiętna 165, zapisana w kodzie szesnastkowym to A5h. Dana ta, powinna być dołączona do instrukcji jako liczba dziesiętna lub w postaci 0A5h ponieważ oznaczenie A5h wskazuje na nazwę symboliczną a nie wartość liczbową (znak litery na pierwszej pozycji).

Komentarz opisuje wykonywaną przez rozkaz czynność i musi być poprzedzony znakiem średnika. Pomimo tego, że sama instrukcja informuje o tym, co

będzie zrobione, dodatkowy komentarz zdecydowanie poprawia zrozumienie działania programu - szczególnie, gdy opis ten zawiera informację rozszerzoną na inne linie programu. W procesie kompilacji, treść komentarza jest pomijana. Pojęcie komentarza wiąże się nie tylko z poszczególnymi liniami programu. Jest dobrym zwyczajem, by komentowany był również zestaw rozkazów, przeznaczony do wypełnienia określonych zadań np. podprogram obsługi zdarzenia. Taki komentarz umieszcza się zwykle na początku opisywanego podprogramu (patrz na przykład programu, podany na końcu tego podrozdziału).

Instrukcje własne i kontrolki asemblera.

Oprócz instrukcji powiązanych z konkretnym typem mikrokontrolera, asembler używa instrukcji własnych i tzw. kontrolki (ang. assembler controls), które bardzo ułatwiają tworzenie programu. Instrukcje te nazywane są dyrektywami asemblera. Do instrukcji własnych asemblera należą, np. dyrektywy *EQU*, *ORG*, *DB* i inne. Dyrektywy te są traktowane podobnie do instrukcji mikrokontrolera i mogą być wstawiane do kodu źródłowego. Dyrektywy nie są instrukcjami mikrokontrolera ale są instrukcjami asemblera - dzięki nim kompilator otrzymuje wskazówki, jak wykreować kod wynikowy (często nazywane są pseudoinstrukcjami). Przykładowo: dyrektywa *ORG adres* (ang. organization) powoduje, że następujące po dyrektywie instrukcje są lokowane w pamięci programu od adresu wskazanego przez tę dyrektywę. Dyrektywa *EQU* (ang. equal) jest używana do określania wartości liczbowej nazw symbolicznych. Wykonanie dyrektywy: *nazwa EQU wartość* powoduje, że do słowa *nazwa* zostanie przypisana dana o konkretnej wartości liczbowej. Od tego momentu, wszystkie instrukcje, które chcą odwołać się do tej danej, mogą to uczynić przez podawanie jej nazwy. Dyrektywa *DB* pozwala ulokować w pamięci programu bajt lub grupę bajtów. Dyrektywa *END*, będąca informacją o końcu programu, powinna być umieszczona w miejscu końca treści programu - dalszy tekst pliku programu nie jest poddawany kompilacji i może być traktowany jako "brudnopis" programisty.

Dyrektywy asemblera, nazywane kontrolkami, przyspieszają proces kompilacji i generowania pliku listingu (plik z rozszerzeniem *LST*). Kontrolki mają ściśle określone nazwy własne, poprzedzane znakiem "\$". Kontrolki, nazywane podstawowymi, mają wpływ na proces kompilacji i muszą być umieszczane na początku programu. Inne, nazywane ogólnymi, mogą być umieszczane w dowolnym miejscu programu. Przykładowo (patrz podany niżej przykład programu): kontrolka podstawowa *\$NOMOD51* powoduje dezaktywację symboliki strefy SFR, która w momencie uruchomienia programu ASEM.W, jest przypisana mikrokontrolerowi 8051. Następną kontrolką, *\$INCLUDE (89S8253.MCU)*, powoduje dopisanie do kodu źródłowego treści pliku, którego nazwa jest objęta nawiasem. Dzięki temu, do kodu programu zostanie wprowadzona symbolika strefy SFR dla mikrokontrolera AT89C8253. Kontrolki ogólne, np. *\$NOLIST* i *\$LIST*, które mogą być umieszczane w dowolnym miejscu tekstu programu powodują, że w pliku listingu wybrane fragmenty informacji o kodzie wynikowym zostaną pominięte.

Bardzo ważną właściwością asemblera jest możliwość zlecenia mu wykonywania operacji arytmetyczno-logicznych. Działania te są wprowadzane do kodu programu przez tzw. operatory a ich wykonywanie odbywa się w trakcie kompilacji kodu źródłowego. Dzięki temu, programista nie musi dokonywać dodatkowych obliczeń - zrobi to za niego asembler. Zastosowanie operatorów pokazano w podanym niżej przykładzie programu (program 1.2.1). W linii 4 programu, przez operację odejmowania, jest definiowana liczba 16-bitowa. Liczba ta, w liniach 15 i 16 ulega rozłożeniu na 2 bajty - młodszy bajt jest obliczany operacją modulo (linia 15) a starszy bajt, za pośrednictwem operacji dzielenia (linia 16). Identyczne działanie można osiągnąć przez zastosowanie innych operatorów: w liniach 29 i 30 uzyskano ten sam wynik przez zastosowanie operatorów *low* i *high*. Inny przykład zastosowania operatora pokazany jest w linii 59.

Przykład programu.

Pokazany niżej przykład programu jest napisany w języku asemblera mikrokontrolera rodziny MCS-51. Program pozwala na odbiór i nadawanie bajtu przez port transmisji szeregowej - natychmiast, po odebraniu znaku małej litery kodu ASCII, jest wysyłany duży znak tej samej litery. Ponadto, co 100ms, zmieniany jest stan świecenia diody LED, dołączonej do mikrokontrolera.

Program 1.2.1

```

$NOMOD51                ; dezaktywuj symbolikę SFR (8051)           (1)
$INCLUDE (89S8253.MCU)   ; dołącz symbolikę SFR kontrolera 89S8253 (2)
$NOLIST                  ; wyłącz listowanie                       (3)
;      stałe programowe i adresy w polu wewnętrznym RAM
t0_dat    equ    65535-921    ; dana dla licznika T0 (przerw. co 1ms)   (4)
timer_buf equ    17           ; adres licznika zdarzeń (od T0)         (5)
send_buf  equ    19           ; adres bufora dla bajtu nadawanego     (6)
stos      equ    60           ; adres początku stosu                  (7)
;      bity flagowe (adres. bajtu RAM: 32)
rec_flag  bit    00h          ; flaga odebrania bajtu przez port szer. (8)
send_flag bit    01h          ; flaga nakazu nadawania przez port szer. (9)
t0_flag   bit    02h          ; flaga wystąpienia przerwania (od T0)   (10)
;      S T A R T   P R O G R A M U
$LIST                    ; włącz listowanie                       (11)
          org    0             ; ustal bieżący adresu kodu (początek)   (12)
inicjacja:  ljmp  start        ; skocz do początku programu            (13)
;      procedury obsługi przerwania
          org    0bh           ; ustal adres proc. przerw. licznika T0  (14)
t0_int:    orl    t0, #t0_dat mod 256    ; (15)
          mov    th0, #t0_dat / 256      ; ustaw rejestry licznika T0             (16)
          setb   t0_flag                 ; ustaw flagę przejścia przez przerwanie (17)
          reti                                ; (18)

```

```

sio_int:    org    23h          ; ustal adres proc. przerw. portu szeregow. (19)
           jbc    ti, sint_20   ; nie rób nic, gdy pusty bufor nadajnika (20)
           clr    ri           ; zeruj flagę odbioru bajtu (21)
           setb   rec_flag      ; ustaw flagę odebrania bajtu (22)
sint_20:    reti              (23)

;          PRZYGOTOWANIE ŚRODOWISKA

           org    0100h        ; ustal bieżący adresu kodu (24)
start:     mov    sp,#stos     ; ustal adres początku stosu (25)

;          ustawienie rejestrów kontrolnych

           mov    pcon,#80h     ; zegar dla sio: taktowanie T1 (19200 b/s) (26)
           mov    scon,#01010000b ; ustawienie parametrów transmisji: (27)
                                     ; tryb 1: 8 bitów, szybkość: T1
           mov    tmod,#00100001b ; ustalenie T1 w tryb 2; T0 w tryb 1; (28)
           mov    tl0,#low t0_da6 ; ustawienie młodszego (29)
           mov    th0,#high t0_dat ; i starszego bajtu licznika T0 (30)
           mov    tl1,#0fdh     ; ustawienie młodszego i starszego bajtu (31)
           mov    th1,#0fdh     ; licznika T1 (19200 b/s) (32)

;          inne ustawienia

           clr    send_flag     ; kasuj flagę gotowości nadajnika (33)
           clr    rec_flag      ; kasuj flagę gotowości odbiornika (34)
           mov    timer_buf,#100 ; ustaw licznik czasomierza progr. (100ms) (35)
           clr    t0_flag       ; zeruj flagę przerwania t0_int (36)
           setb   et0           ; aktywuj przerwanie licznika T0 (37)
           setb   es            ; aktywuj przerwanie portu szeregowego (38)
           setb   ea            ; aktywuj wszystkie przerwania (39)
           setb   tr0           ; uruchom licznik T0 (40)
           setb   tr1           ; uruchom licznik T1 (41)

;          PĘTLA PROGRAMOWA

ptl:
;          testowanie: czy odebrano bajt przez port szeregowy
           jnb    rec_flag,ptl_10 ; dalej gdy brak odbioru RS23 (42)
           clr    rec_flag      ; kasuj flagę odebrania bajtu (43)
           lcall  rec_serv      ; wykonaj obsługę odbioru bajtu (44)

;          testowanie: czy trzeba wysłać bajt przez port szeregowy
ptl_10:    jnb    send_flag,ptl_20 ; dalej, gdy nie ma nic do wysłania (45)
           lcall  send_serv     ; wykonaj obsługę nadawania (46)

;          testowanie: czy licznik T0 zakończył odliczanie (co 1ms)
ptl_20:    jnb    t0_flag,ptl_30 ; dalej gdy brak przerwania od licznika T0 (47)
           clr    t0_flag       ; zeruj flagę (48)
           lcall  t0_serv       ; dokończ obsługę przerwania od T0 (49)
ptl_30:    ljmp   ptl          ; wróć do początku pętli (50)

;          BŁOK PODPROGRAMÓW OBSŁUGI ZDARZEŃ

```

```

;      dokończenie obsługi przerwania od czasomierza T0 (zdarzenie co 1 ms)
;      opis: co 1 ms licznik T0 generuje przerwanie - w ramach dokończenia obsługi
;      tego zdarzenia, zmniejszany jest stan licznika zdarzeń. Po osiągnięciu zera,
;      stan licznika jest regenerowany i zmieniany jest stan diody wskaźnikowej LED
;      (co 100ms).
t0_serv:  mov    a,timer_buf    ; czytaj stan licznika zdarzeń (od T0)      (51)
          jz     ts_10         ; dalej gdy stan=0                          (52)
          dec   timer_buf     ; zmniejsz stan licznika                    (53)
          ret                                     (54)
ts_10:    mov    timer_buf,#100 ; regeneruj stan licznik (100ms)      (55)
          cpl   p1.7          ; zmień stan diody LED (DSM-51)          (56)
          ret                                     (57)

;      dokończenie obsługi przerwania od odbiornika portu szeregowego
;      opis: po odebraniu i skompletowaniu bajtu przez port szeregowy jest
;      generowane przerwanie - w ramach dokończenia obsługi tego zdarzenia,
;      odebrany bajt jest modyfikowany (zamiana małych liter na duże)
;      i przekazywany do wysłania przez port szeregowy .
rec_serv: mov    a,sbuf        ; pobierz bajt z portu szeregowego      (58)
          add   a,#256 - 32    ; zamień kod litery z "małej" na "dużą"  (59)
          mov   send_buf,a     ; i zapamiętaj w buforze pomocniczym    (60)
          setb  send_flag      ; ustaw flagę gotowości do nadawania     (61)
          ret                                     (62)

;      obsługa nadawania przez port szeregowy
;      opis: procedura testuje czy możliwe jest wysłanie bajtu - gdy wysyłanie
;      jest możliwe to flaga zlecenia wysłania jest kasowana a bajt wysyłany.
send_serv: jnb   ti,se_10     ; dalej, gdy nadajnik jest gotowy do nadaw. (63)
          ret                                     (64)
se_10:    clr   send_flag     ; zeruj flagę nadawania bajtu          (65)
          mov   a,send_buf     ; pobierz bajt do wysłania            (66)
          mov   sbuf,a         ; i wyślij go                          (67)
          ret                                     (68)
end                                              (69)

```

Struktura pokazanego wyżej programu zawiera elementy, na które zwrócono uwagę w poprzednich rozdziałach części 2 skryptu. Program posiada element przygotowania środowiska do pracy w pętli i samą pętlę. Przygotowanie środowiska jest wykonywane przez linie programu, od 24 do 41. Pętla programowa jest rozpięta pomiędzy liniami, od 42 do 50. W celu poprawienia czytelności programu pętli, obsługa poszczególnych zdarzeń została przeniesiona do oddzielnego bloku, który nazwano *blokiem podprogramów obsługi zdarzeń*. Podprogramy te zgrupowano pomiędzy liniami, od 51 do 69. Do podprogramów obsługi zdarzeń należą również procedury przerwaniowe, które w omawianym przykładzie programu umieszczono pomiędzy liniami, od 14 do 23. Oczywiście, procedury te mogą być umieszczone w bloku obsługi zdarzeń ale, ze względu na naturalne ich położenie w początkowej części pamięci programu, pozostawiono je tam gdzie są.

Po przygotowaniu środowiska, w pętli programowej jest wykonywane sukcesywne przepytanie stanu środowiska (badanie stanu flag poszczególnych zdarzeń) a obsługa zdarzeń jest realizowana przez wywoływanie przypisanych zdarzeniom podprogramów. W programie, ze względu na jego "mikroskopijny" rozmiar, nie występuje problem zdarzeń powiązanych, nie ma problemu obsługi zdarzeń podprogramami długimi. Udało się jednak pokazać rozdzielanie obsługi zdarzeń na elementy, w przypadku zgłaszania ich przez przerwania. Przerwanie jest generowane, np. w momencie przepełnienia się licznika T0. W ramach procedury przerwaniowej wykonywana jest najważniejsza czynność: przeładowanie rejestrów licznika (linie 15 i 16 oraz pomocniczo, linia 17). Pozostałe elementy obsługi są realizowane w obszarze pętli (linie 47..49) oraz podprogram obsługi zdarzenia (linie: 51..57).

1.2.2. Programowanie w języku C.

Tradycyjnie systemy mikroprocesorowe programowane są w assemblerze. Od pewnego czasu obserwuje się jednak tendencję wykorzystywania do tego celu języków wyższego poziomu, w szczególności języka C. Przyczyny tego zjawiska są różne, a z najważniejszych należy wymienić:

- uważa się, że język C jest łatwiejszy od assemblera, jest zestandaryzowany, posiada bogatą literaturę;
- czas potrzebny na napisanie i uruchomienie programu w języku C powinien być krótszy niż analogiczny czas dla assemblera – dotyczy to zwłaszcza większych projektów;
- język C jest na tyle niskopoziomowy, że dobrze kompiluje się do kodu maszynowego, a równocześnie na tyle wysokopoziomowy, że umożliwia programowanie strukturalne;
- strukturalny charakter języka ułatwia tworzenie fragmentów kodu możliwych do wielokrotnego wykorzystania i poprawiających wykorzystanie pamięci programu;
- starannie napisany program w języku C jest łatwiejszy do zrozumienia, analizy i konserwacji;
- przenoszenie programu w języku C z jednego typu mikrokontrolera na inny jest łatwiejsze, wymaga mniejszych zmian w kodzie, zajmuje mniej czasu,
- pojawia się coraz więcej narzędzi (kompilatory, debuggery, symulatory) umożliwiających bądź ułatwiających tworzenie oprogramowania w ten sposób;
- wydajność współcześnie produkowanych mikrokontrolerów jest na tyle wysoka, że jest w stanie skompensować nieco niższą wydajność programów napisanych w języku C, w porównaniu z programami napisanymi w assemblerze.

Pomimo przedstawionych zalet, programowanie w językach wysokiego poziomu nie zwalnia twórców od szczegółowego poznania architektury systemu mikrokomputerowego i technik programistycznych, zapewniających powstanie poprawnego programu. Bez zrozumienia natury narzędzi i należytej staranności podczas programowania łatwo jest popełnić błędy, które nie skutkują komunikatami kompilatora, ale co do których można mieć poważne zastrzeżenia. Jako przykłady można podać rozrzutne gospodarowanie pamięcią albo nieefektywne wykorzystanie czasu mikroprocesora.

Kompilator

Na potrzeby zajęć laboratoryjnych wybrany został kompilator SDCC (ang. Small Device C Compiler), którego autorem jest Sandeep Dutta [2]. Jest to kompilator dostępny na zasadach GNU (ang. General Public License). Taki wybór podyktowały następujące cechy kompilatora:

- umożliwia zrozumienie pełnego cyklu generowania programu dla mikrokontrolera;
- dostarczany jest z debuggerem i symulatorem;
- generuje kod dla mikrokontrolerów rodzin: MCS51, Dallas DS80C390, Motorola HC08, Zilog Z80 (inne w przygotowaniu);
- zmiana docelowego mikrokontrolera nie wymaga istotnych zmian w kodzie źródłowym;
- dostępne są wersje dla systemów operacyjnych Windows i Linux;
- może być legalnie wykorzystywany do tworzenia oprogramowania (również komercyjnego) bez ponoszenia opłat, z zapewnieniem utrzymania tego sposobu licencjonowania.

Kompilator dokonuje optymalizacji kodu między innymi poprzez:

- wykrywanie nie używanych zmiennych;
- eliminowanie nigdy nie wykonywanych fragmentów kodu;
- tworzenie tablicy skoków dla instrukcji *switch*.

Ponadto kompilator wspiera pracę programisty obsługując *inline assembler* i oferując możliwość identyfikowania funkcji o potencjalnie zbyt dużej złożoności – można je napisać ponownie w sposób bardziej optymalny, a w razie potrzeby zakodować w asemblerze.

Wartości liczbowe w kodzie źródłowym mogą być podawane w formatach przedstawionych w tabeli 1.2.1 (wszystkie przedstawione w tabeli przykłady reprezentują dziesiętnie wartość 65).

Tabela 1.2.1. Format zapisu wartości liczbowych dla kompilatora SDCC.

format	przykład	uwagi
binarny	0b01000001	prefiks 0b
ósemkowy	0101	prefiks 0
szesnastkowy	0x41	prefiks 0x
znakowy	'A'	w apostrofach
dziesiętny	65	bez prefiksu

Na etapie kompilacji mogą być wykonywane podstawowe operacje arytmetyczne, dzięki czemu w kodzie wynikowym zamiast 'a' - 'A' w zapisie

```
c += 'a' - 'A';
```

pojawi się wyliczona wartość 20h, podczas gdy całość łatwiej jest zrozumieć jako konwersję litery dużej na małą.

Typy danych wspierane przez kompilator przedstawia tabela 1.2.2.

Tabela 1.2.2. Typy danych wspierane przez kompilator SDCC.

typ	rozmiar	domyślnie	zakres	
			ze znakiem	bez znaku
bool	1 bit	unsigned	-	0, 1
char	8 bits, 1bytes	signed	-128.. 127	0.. 255
short	16 bits, 2 bytes	signed	-32 768.. 32 767	0.. 65 535
int	16 bits, 2 bytes	signed	-32 768.. 32 767	0.. 65 535
long	32 bits, 4 bytes	signed	-2 147 483 648 ..+2 147 483 647	0.. 4 294 967 295
float	4 bytes	signed		1.175494351E-38 ..3.402823466E+38
pointer	1, 2, 3 or 4 bytes			

Ogólne zasady redagowania kodu źródłowego programu.

Tak jak w przypadku assemblera i znakomitej większości innych języków programowania, kod źródłowy programów pisanych w języku C musi być zapisany w pliku, w czystym formacie tekstowym (ASCII).

Plik musi mieć rozszerzenie .C albo .H – kompilator SDCC akceptuje wyłącznie takie rozszerzenia.

Tak jak powinno być już wiadome, język C nie jest wrażliwy na formatowanie kodu źródłowego. Nie licząc drobnych wyjątków cały program można zapisać w jednej linijce długości kilku kilobajtów. Jeżeli taki program będzie poprawny formalnie, to zostanie skompilowany, a na jego podstawie powstaną pliki pośrednie i plik wynikowy. Taki sposób redagowania czyni jednak kod źródłowy zupełnie nieczytelny, niepodatny na konserwację i rozbudowę.

Na poprawny, czytelny i estetyczny styl redagowania kodu źródłowego w języku C składają się następujące elementy (za Apache Developers C Language Style Guide [9]):

- Komentarze. Jeżeli funkcja lub fragment kodu nie są trywialne, należy w rozsądny sposób je skomentować.
- Wcięcia. Każdy blok programu powinien być wcięty o taki sam odstęp (na przykład 4 spacje). Zaleca się używanie spacji, a nie tabulatora – nie wystąpią wówczas problemy przy przenoszeniu kodu pomiędzy różnymi edytorami.

- Funkcje powinny być pisane wg szablonu:

```
int main(int argc, char **args)
{
    kod funkcji;
}
```

- Szablon wywołania funkcji:

```
funkcja(a, r, g);
```

(spacje po przecinkach, bez spacji w sąsiedztwie nawiasów otwierającego i zamykającego),

- Instrukcje sterujące przepływem programu (**if**, **while**, **for** itp.) powinny być formatowane wg szablonu:

```
if (wyrażenie) {
    kod;
}
else {
    kod;
}
```

- Szablon instrukcji **for**:

```
for (a; b; c) (spacje po średnikach)
```

- Szablon instrukcji **switch**:

```
switch (x) {
case a:
    kod;
case b:
    kod;
}
```

- Operatory dwuargumentowe powinny być otoczone przez spacje, operatory jednoargumentowe (negacja, inkrementacja, dekrementacja) nie powinny być oddzielane od argumentu spacją:

```
a = b
a + b
a < b
a = -b
a = !b
++a
```

Reguły te nie mają sztywnego charakteru, dopuszcza się rozsądne, ale konsekwentnie stosowane odstępstwa.

Przykład programu.

Ze względów dydaktycznych poniższy program jest niemal kalką programu assemblerowego przedstawionego w poprzednim podrozdziale. Możliwe jest w ten sposób samodzielne porównanie kodów programów i zapoznanie się z technikami realizacji w języku C rozwiązań poznanych już dla assemblera. Program został napisany zgodnie z omówionymi dotychczas zaleceniami.

Program 1.2.2. Przykład programu – język C.

```

1: #define FALSE 0
   : #define TRUE 1
3:
   : #define T0_DAT 65535-921 // przerwanie T0 co 1ms
5: #define TL_0 T0_DAT%256 // tak będzie łatwiej
   : #define TH_0 T0_DAT/256 // przeładować timer
7: #define T100 100 // półokres LED
   :
9: /* -----
   : * MAPOWANIE ZALEŻNE OD TYPU MIKROKONTROLERA
11: *
   : * poniższe mapowanie ma charakter poglądowy, jest
13: * zbędne w przypadku dołączenia pliku nagłówkowego
   : * #include <at89s8252.h>
15: */
   :
17: __sfr __at (0x87) PCON; // power control
   : __sfr __at (0x98) SCON; // serial control
19: __sfr __at (0x89) TMOD; // timer mode
   :
21: __sfr __at (0x8C) TH0; // starszy bajt timer'a t0
   : __sfr __at (0x8A) TL0; // młodszy "-"
23: __sfr __at (0x8D) TH1; // starszy bajt timer'a t1
   : __sfr __at (0x8B) TL1; // młodszy "-"
25:
   : __sfr __at (0x99) SBUF; //bufor nad./odb. UART
27:
   : __sbit __at (0x99) TI; //flaga koniec nad. UART
29: __sbit __at (0x98) RI; //flaga koniec odb. UART
   :
31: __sbit __at (0xA9) ET0; // aktywność przerwania T0
   : __sbit __at (0xAC) ES; // aktywność przerwania UART
33: __sbit __at (0xAF) EA; // zezwolenie obsługi przerwania
   : __sbit __at (0x8C) TR0; // aktywność licznika T0
35: __sbit __at (0x8E) TR1; // aktywność licznika T1
   :
37: /* KONIEC MAPOWANIA
   : * ----- */
39:
   : //zmienne globalne

```

```

41: unsigned char timer_buf; // czasomierz programowy
   : unsigned char send_buf; // bufor bajtu nadawanego
43:
   : //flagi bitowe
45: __bit __at (0x97) LED; // bit 7 portu P1 sterujący LED
   : __bit rec_flag; // flaga odebrania znaku
47: __bit send_flag; // dane gotowe do transmisji
   : __bit t0_flag; // flaga przerwania licznika T0
49:
   : //deklaracje funkcji
51: void rec_serv(void);
   : void send_serv(void);
53: void t0_serv(void);
   :
55: /*-----*
   : * START PROGRAMU *
57: *-----*/
   :
59: void main()
   : {
61: //SDCC generuje kod ustawiający Stack Pointer
   :
63: // PRZYGOTOWANIE ŚRODOWISKA
   : //-----
65:
   : //ustawienie rejestrów kontrolnych
67: //-----
   :
69: PCON = 0x80; // zegar dla sio, T1 (19200 b/s)
   : SCON = 0b01010000; //ustaw parametry transmisji
71: //tryb 1: 8 bitów, szybkość: T1
   : TMOD = 0b00100001; //ustaw T1 w tryb 2; T0 w tryb 1
73:
   : TL0 = TL_0; //ustawienie młodszego i starszego
75: TH0 = TH_0; //bajtu T0 przerwanie co 1 milisekundę
   :
77: TL1 = 0xFD; //ustawienie młodszego
   : TH1 = 0xFD; //i starszego bajtu T1 (19200)
79:
   : //inne ustawienia
81: //-----
   : timer_buf = T100; // ładuj timeout T0 (100ms)
83: send_flag = FALSE; // kasuj flagę gotowości danych
   : rec_flag = FALSE; // kasuj flagę odbiornik gotowy
85: t0_flag = FALSE; // zeruj flagę przerw. t0_int
   :
87: ET0 = TRUE; // aktywuj przerwanie od licznika T0
   : ES = TRUE; // aktywuj przerwanie od UART
89: EA = TRUE; // aktywuj wszystkie przerwania
   : TR0 = TRUE; // uruchom licznik T0
91: TR1 = TRUE; // uruchom licznik T1
   :
93: // PĘTLA PROGRAMOWA
   : //-----
95: while (TRUE) {

```

```

:
97:         if (rec_flag) {           //odebrany bajt w buf. UART
:             rec_flag = FALSE; //kasuj flagę bajt odebrany
99:             rec_serv();           //obsłuż odebrany bajt
:         }
101:
:         if (send_flag)             //trzeba wysłać dane UART
103:             send_serv();           //wykonaj obsługę nadawania
:
105:         //podczas ostatniego obrotu pętli wystąpiło
:         if (t0_flag) {             //przerwanie zegarowe
107:             t0_flag = FALSE; //zeruj flagę
:             t0_serv();             //obsłuż przerwanie od T0
109:         }
:     }
111: }
:
113: /* ----- *
:  *   BLOK PODPROGRAMÓW OBSŁUGI ZDARZEŃ   *
115: * ----- */
:
117: /*
:  * dokończenie obsługi przerwania czasomierza T0, co 1 ms
119: * opis: co 1 ms licznik T0 generuje przerwanie, w ramach
:  * dokończenia obsługi tego zdarzenia, zmniejszany jest
121: * stan licznika zdarzeń. Po wyzerowaniu licznik jest
:  * regenerowany i zmieniany jest stan diody LED, co 100ms
123: */
: void t0_serv(void)
125: {
:     if (timer_buf)
127:         timer_buf--;             //zmniejsz stan czasomierza
:     else {
129:         timer_buf = T100;         //regeneruj licznik (100ms)
:         LED = !LED;              //zmień stan diody LED
131:     }
: }
133:
: /*
135: * dokończenie obsługi przerwania od odbiornika UART
: * opis: po odebraniu i skompletowaniu bajtu przez port
137: * szeregowy jest generowane przerwanie - w ramach
: * dokończenia obsługi tego zdarzenia, odebrany bajt jest
139: * modyfikowany (zamiana małych liter na duże)
: * i przekazywany do wysłania przez port szeregowy
141: */
: void rec_serv(void)
143: {
:     unsigned char uc = SBUF; //pobierz z bufara RS'a
145:     if ( ( uc >= 'a' ) && ( uc < 'z' + 1 ) )
:         uc += 'A' - 'a'; //zamień małą na wielką
147:
:     send_buf = uc;              //zapamiętaj w buforze
149:     send_flag = TRUE;           //ustaw flagę gotowości danych
: }

```

```

151:
: /*
153: * obsługa nadawania przez port szeregowy
: * opis: procedura testuje czy można wysłać bajt
155: * jeżeli tak to flaga zlecenia wysłania jest
: * kasowana a bajt wysyłany.
157: */
: void send_serv(void)
159: {
:     if (TI) //nadajnik nie jest gotowy
161:         return;
:
163:     send_flag = FALSE;    //zeruj flagę nadawania bajtu
:     SBUF = send_buf;     //wyslij bajt
165: }
:
167: /* ----- *
: *   PROCEDURY OBSŁUGI PRZERWAŃ *
169: * ----- */
:
171: /*
: * przerwanie #1 od timer'a 0
173: * wywoływane przez LJMP spod adresu 0x000b
: */
175: void t0_int(void) __interrupt(1)
: {
177:     TL0 = TL0 | TL_0;    //odświeża licznik T0
:     TH0 = TH_0;        //ustawia flagę sygnalizującą
179:     t0_flag = TRUE;     //fakt wystąpienia przerwania
: }
181:
: /*
183: * przerwanie #4 od UART
: * wywoływane przez LJMP spod adresu 0x0023
185: */
: void sio_int(void) __interrupt(4)
187: {
:     if (TI) {           //gdy pusty bufor nadajnika
189:         TI = FALSE;
:     }
191:     else {
:         RI = FALSE;    //zeruj flagę odbioru znaku
193:         rec_flag = TRUE; //ustaw flagę odebrania znaku
:     }
195: }
:

```

W wierszach od 1 do 7, dyrektywami **#define** preprocesora, zadeklarowano w sposób typowy dla języka C stałe, wykorzystywane wielokrotnie w programie.

W wierszach od 17 do 35 zademonstrowano poglądowo mapowanie rejestrów SFR mikrokontrolera. W typowym programie mapowanie uzyskuje się

poprzez dołączenie dyrektywą `#include` preprocesora odpowiedniego pliku nagłówkowego, na przykład:

```
#include <at89s8252.h>
```

W wierszach od 41 do 53 zawarto jeszcze kilka deklaracji o zasięgu globalnym.

Kompilator automatycznie wyznacza i generuje kod ustawiający początkową wartość wskaźnika stosu. W przypadku programowania w asemblerze pominięcie tej ważnej czynności jest często przyczyną źle działającego lub w ogóle nie działającego programu. W przykładowym programie kompilator wygenerował wartość `3Ch` jako adres początkowy stosu. Oznacza to, że pierwsze wywołanie funkcji umieści adres powrotu pod adresami `3Dh` i `3Eh` (odpowiednio młodszy i starszy bajt).

Wykonywanie programu rozpoczyna ustalenie pożądaných parametrów początkowych środowiska (wiersze 69 do 91).

Pętla programowa (wiersze 95 do 110) obserwuje i reaguje na 3 zmiany w środowisku, sygnalizowane stanem bitów flagowych:

1. odebrano znak z portu szeregowego,
2. należy wysłać znak przez port szeregowy,
3. podczas ostatniego obrotu pętli miało miejsce przerwanie zegarowe (upłynęła kolejna milisekunda)

Bity flagowe ustawiane są przez procedury przerwaniowe. Dla bitów `rec_flag` i `t0_flag` zależność jest prosta. Dla bitu `send_flag` zależność jest nieco bardziej złożona, ale i tak łatwa do wyśledzenia na podstawie kodu źródłowego.

W przypadku stwierdzenia zmian w środowisku, wywoływane są funkcje `void t0_serv(void)`, `void rec_serv(void)` i `void send_serv(void)` odpowiednio reagujące na te zmiany. Kod funkcji jest prosty i nie będzie omawiany.

Pusty wiersz 196, kończący kod źródłowy, jest elementem wymaganym składniowo przez język C.

Funkcje obsługujące przerwania

Zagadnienie kodowania funkcji przerwaniowych wymaga nieco uwagi.

W przypadku programu w asemblerze, poprawną obsługę przerwania zapewnia umieszczenie kodu funkcji pod odpowiednim adresem, wynikającym z architektury mikrokontrolera. Na przykład dla mikrokontrolera rodziny 80C51 i dla przerwania od timer'a 0 jest to adres `000Bh` (dla timer'a 1 – `001Bh`), a dla przerwania od układu transmisji szeregowej adres `0023h`. Pod wskazanymi adresami (również pod adresami przewidzianymi dla innych przerwań) w architekturze 80C51 dostępnych jest 8 bajtów na kod realizujący funkcję przerwaniową. W wielu przypadkach jest to ilość wystarczająca. Jeżeli jednak okazałyby się

zbyt mała, można wykorzystać kod znajdujący się w innym obszarze pamięci wykonując instrukcję skoku `1jmp`.

W przypadku programów kompilowanych kompilatorem SDCC pod adresem ładowanym do rejestru PC w przypadku wystąpienia przerwania, umieszczana jest od razu instrukcja skoku `1jmp` do funkcji przerwaniowej.

Nagłówek funkcji przerwaniowej może wyglądać dwójako:

```
void t0_int(void) __interrupt(1)
```

albo

```
void t0_int(void) __interrupt(1) __using(1)
```

Kompilator rozpoznaje funkcje przerwaniowe po słowie kluczowym `__interrupt`, a towarzyszący mu argument (w przykładzie `(1)`) identyfikuje jedno z pięciu przerw. Kompilator wiąże numer przerwania (na przykład timer 0 generuje przerwanie pierwsze) z odpowiednim dla mikrokontrolera adresem procedury obsługi przerwania.

Użycie słowa kluczowego `__using(n)` spowoduje wygenerowanie kodu zabezpieczającego rejestr PSW na stosie i wybierającego odpowiedni (`n`) bank rejestrów R. Przed zakończeniem funkcji przerwaniowej rejestr PSW zostanie odzyskany ze stosu, a tym samym zostanie przywrócony poprzedni bank rejestrów R.

Stos

Należy pamiętać, że obszar pamięci RAM przewidziany do przechowywania zmiennych (dla architektury 80C51 są to adresy od 30h do 7Fh – łącznie 80 bajtów) jest również wykorzystywany do przechowywania zawartości stosu. Każde wywołanie funkcji generuje instrukcję `1call` asemblera, a ta odkłada na stosie adres powrotu. Przekazywanie argumentów wywołania funkcji odbywa się przez stos. W przypadku wplatania kodu *inline assembler* instrukcja `push` również korzysta ze stosu. Łatwo wywnioskować, że próby wykorzystania źle zaprojektowanych funkcji rekurencyjnych mogą szybko doprowadzić do przepełnienia stosu.

Prezentowany przykładowy program (1.2.2) operuje na zmiennych globalnych. W ten sposób unikamy wykorzystania stosu na potrzeby przekazywania argumentów do wywoływanych funkcji.

Podobnie bardzo racjonalnie odbywa się wywoływanie funkcji. Jedynym miejscem, gdzie jawnie wywoływane są jakiekolwiek funkcje jest pętla programowa, przy czym wywołanie następnej funkcji nastąpi dopiero po zakończeniu wykonywania poprzedniej.

Analizując obciążenie stosu w przykładowym programie widać, że najbardziej niekorzystny przypadek ma miejsce wtedy, gdy podczas wykonywania funkcji wysyłającej znak przez port szeregowy (`void send_serv(void)`) wystąpi przerwanie wywołane odebraniem znaku przez port szeregowy, a podczas

obsługi tego przerwania (`void sio_int(void)`) wystąpi przerwanie zegarowe, które ma wyższy priorytet i wywoła funkcję `void t0_int(void)`. Na stosie zostanie wówczas odłożone 6 bajtów trzech adresów powrotnych.

Dobłą praktyką programistyczną jest przeanalizowanie, czy dla najbardziej niekorzystnego przypadku wskaźnik stosu nie przyjmie niedopuszczalnych wartości.

Obsługa pamięci zewnętrznej i urządzeń wejścia/wyjścia

W przypadku mikrokontrolerów rodziny 80C51 obsługa pamięci zewnętrznej i urządzeń wejścia/wyjścia realizowana jest instrukcją `movx` asemblera. Poniżej zostaną omówione techniki dostępu do wymienionych zasobów na przykładzie klawiatury matrycowej systemu DSM-51.

Klawiatura matrycowa systemu DSM-51 składa się z 16 przycisków [3]. Funkcjonują one jako dwa urządzenia wejściowe o adresach 0FF21h (przyciski 0.. 7) i 0FF22h (pozostałe 8 przycisków).

Odczyt stanu klawiatury w przypadku programu w asemblerze pokazuje program 1.2.3.

Program 1.2.3. Odczyt stanu klawiatury – asembler.

```
1: mov    dptr,#0FF21h    ;adres klawiatury do DPTR
2: movx   a,@dptr        ;stan klawiatury do ACC
```

W przypadku programu w języku C można wykorzystać fakt, że kompilator SDCC obsługuje tzw. *inline assembler* (program 1.2.4). W przykładzie takiego rozwiązania należy zauważyć, że zmienna `key`, do której odwołuje się kod asemblera, została zadeklarowana jako globalna (wiersz 2) oraz że w kodzie asemblera nazwa zmiennej poprzedzona jest znakiem podkreślenia (`_key` w wierszu 14).

Program 1.2.4. Odczyt stanu klawiatury – inline assembler.

```
1: ...
   : unsigned char key;
3: ...
   : ...
5: _asm                               ;dalej kod asemblera
   : push    dpl                       ;zabezpieczamy rejestry
7: push    dph
   : push    psw
9: push    acc
   :
11: mov    dptr, #0FF21h                ;adres klawiatury do dptr
   : movx   a, @dptr                    ;stan klawiatury do akumulatora
13:                               ;dalej program w języku C
   : mov    _key, a                      ;odczyta stan klawiatury
15:                               ;ze zmiennej key
```

```

: pop    acc           ;odzyskujemy rejestry
17: pop psw
: pop    dph
19: pop    dpl
: _endasm;           //koniec kodu asemlera
21: ...

```

W tym konkretnym przypadku wykorzystanie *inline assemble* jest dopuszczalne, ponieważ obsługa urządzeń wejścia/wyjścia jest ściśle powiązana z konkretnym rozwiązaniem sprzętowym i zagadnienie przenośności kodu schodzi na plan dalszy.

Możliwe jest jednak takie przygotowanie kodu w języku C, że kompilator SDCC sam wygeneruje pożądaną instrukcję `movx`, z odpowiednimi wartościami załadowanymi do rejestru `dptr` (`dpl` i `dph`). Pokazuje to program 1.2.5.

Program 1.2.5. Odczyt stanu klawiatury – język C.

```

1: void odczyt_klawiatury()
: {
3:     //wskaźnik na adres 0xFF21 w obszarze xdata
:     __xdata unsigned char * key_addr =
5:         (__xdata unsigned char *) 0xFF21;
:
7:     //zmienna przyjmująca stan klawiatury
:     unsigned char key;
9:
:     //kompilator wygeneruje movx ładując 0xFF21 do dptr
11:    key = *key_addr;
:
13:    /*
:     ....
15:    */
: }

```

Dostęp do pamięci programu

Stałe elementy programu są powszechnie przechowywane razem z kodem programu w pamięci programu. Programując w asemblerze deklarujemy takie elementy mnemonikiem `DB` i uzyskujemy do nich dostęp instrukcją `movc`.

Programując w języku C można poprzedzić deklaracje "zmiennych" słowem kluczowym `__code` i osiągnąć taki sam rezultat. Zmienne deklarowane w ten sposób zostaną umieszczone w pamięci programu, a kompilator wygeneruje instrukcję `movc` aby odczytać ich wartość. Próbę zmiany wartości takich zmiennych kompilator potraktuje jako błąd.

Klasycznym przykładem wykorzystania pamięci programu do przechowywania stałych elementów programu są szablony (wzory) cyfr dla siedmiosegmentowego, multipleksowanego wyświetlacza LED (dalej wyświetlacza LED).

Zamieszczony poniżej kod programu jest przykładem obsługi wyświetlacza LED. Program ma 2 zadania:

- uzupełnia program 1.2.5, pokazując jak wykonać operację zapisu do urządzenia wejścia/wyjścia albo pamięci zewnętrznej,
- pokazuje jak spowodować, by stałe elementy programu zostały umieszczone w pamięci programu bez uszczuplania dostępnej pamięci RAM.

Program 1.2.6 został bardzo uproszczony i ma charakter poglądowy.

Program 1.2.6. Dostęp do pamięci programu.

```

1: #define TRUE 1
   : #define FALSE 0
3:
   : //szablony cyfr (od 0 do 9) dla wyświetlacza LED
5: //przechowywane w pamięci programu (__code)
   : __code unsigned char WZOR[10] = { 0b01111111, 0b00001110,
7:           0b10110111, 0b10011111, 0b11001110, 0b11011011,
   :           0b11111011, 0b00001111, 0b11111111, 0b11011111 };
9:
   : //bit 6 portu 1 włącza/wyłącza wyświetlacz LED
11: __bit __at (0x96) SEG_OFF;
   :
13: void main()
   : {
15:     //bufor wybierający bitowo aktywny wyświetlacz
   :     __xdata unsigned char * led_wyb =
17:           (__xdata unsigned char *) 0xFF30;
   :     //bufor wybierający aktywne segmenty wyświetlacza
19:     __xdata unsigned char * led_led =
   :           (__xdata unsigned char *) 0xFF38;
21:
   :     unsigned char led_p, //indeks aktywnego wyświetlacza
23:           led_b; //aktywny wyświetlacz (bitowo)
   :
25:     while (TRUE) { //pętla nieskończona
   :
27:         //pętla przebiegająca po 6 wyświetlaczach
   :         for (led_p = 0, led_b = 1;
29:             led_p < 6;
   :             led_p++, led_b += led_b) {
31:
   :             SEG_OFF = TRUE;           //wyłącza wysw. LED
33:             *led_wyb = led_b;         //wybiera wyświetlacz
   :             *led_led = WZOR[led_p]; //wybiera segmenty
35:             SEG_OFF = FALSE;          //włącza wysw. LED
   :         }
37:     }
   : }
39:

```

- ze względu na deklarację `__code` (wiersz 6) tablica `wzor[10]` (przechowująca szablony cyfr wyświetlacza LED) zostanie umieszczona w pamięci programu, a jej odczyt (wiersz 34) wygeneruje instrukcję `movc` assemblera, pominięcie słowa kluczowego `__code`, spowoduje umieszczenie tej tablicy w pamięci RAM mikrokontrolera, w tym przypadku jest to zupełnie nieuzasadnione,
- zapis pod adres `led_wyb` (wiersz 33) wygeneruje instrukcję `movx` assemblera ze względu na deklarację `__xdata` w wierszu 16,
- zapis pod adres `led_led` (wiersz 34) wygeneruje instrukcję `movx` assemblera ze względu na deklarację `__xdata` w wierszu 19.

Podsumowanie

W niniejszym podrozdziale przedstawione zostały techniki programowania mikrokontrolera z wykorzystaniem języka C i kompilatora SDCC.

W szczególności zaprezentowane zostały:

- dostęp do zasobów mikrokontrolera z wykorzystaniem rejestrów specjalnego przeznaczenia SFR, dotyczy to zarówno dostępu do całego bajtu (na przykład bufor SBUF) jak i pojedynczych bitów (flaga TI);
- idea pętli programowej wykorzystującej w sposób maksymalny czas mikrokontrolera;
- idea wykorzystania zmiennych globalnych i obsługi zdarzeń w pętli programowej bez ryzyka wystąpienia niekontrolowanego zapotrzebowania na pamięć;
- realizacja funkcji przerwaniowych, w tym obsługa portu transmisji szeregowej i sposób odmierzania czasu w oparciu o przerwanie zegarowe;
- zastosowanie wskaźników do obsługi pamięci zewnętrznej i urządzeń wejścia/wyjścia, czyli tam gdzie wymagane jest użycie instrukcji `movx`;
- dostęp do danych przechowywanych w pamięci programu, wymagający użycia instrukcji `movc`.

Jest to wystarczający zasób wiedzy, by bez obaw rozpocząć programowanie mikrokontrolerów w języku C z wykorzystaniem kompilatora SDCC.

CZĘŚĆ 3

UZUPEŁNIENIA

3.1. TABELE I OPISY.....	142
3.1.1. KONTROLER HD44780.....	142
3.1.2. MIKROKONTROLER 80C51/52 I 89S8253.....	144
3.1.3. SYSTEM MIKROPROCESOROWY FTSM_51.....	144
3.2. SKRÓCONA LISTA ROZKAZÓW.	159
3.3. PEŁNA LISTA ROZKAZÓW.....	165

1.3. Tabele i opisy.

1.3.1. Kontroler HD44780.

Tabela 1.3.1. Wykaz znaków wyświetlanych przez kontroler HD44780.

		cztery starsze bity															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
cztery młodsze bity	0000	CG RAM [1]			0	Q	P	`	P				-	9	3	Q	P
	0001	CG RAM [2]		!	1	A	Q	a	9			。	ア	チ	4	3	Q
	0010	CG RAM [3]		"	2	B	R	b	r			「	イ	ツ	×	β	θ
	0011	CG RAM [4]		#	3	C	S	c	s			」	ウ	テ	ε	ε	∞
	0100	CG RAM [5]		\$	4	D	T	d	t			、	エ	ト	ト	μ	Ω
	0101	CG RAM [6]		%	5	E	U	e	u			・	オ	ナ	1	ε	Ü
	0110	CG RAM [7]		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
	0111	CG RAM [8]		'	7	G	W	g	w			ア	キ	ヌ	ラ	Q	π
	1000	CG RAM [1]		<	8	H	X	h	x			イ	ク	ネ	リ	ル	̄
	1001	CG RAM [2])	9	I	Y	i	y			ウ	ケ	ル	ル	、	4
	1010	CG RAM [3]		*	:	J	Z	j	z			エ	コ	ハ	レ	i	≠
	1011	CG RAM [4]		+	;	K	[k	[オ	サ	ヒ	ロ	*	π
	1100	CG RAM [5]		,	<	L	¥	l	l			ヤ	シ	フ	ワ	φ	π
	1101	CG RAM [6]		-	=	M]	m]			ユ	ズ	ハ	ン	も	÷
	1110	CG RAM [7]		.	>	N	^	n	→			ヨ	セ	ホ	、	ñ	
	1111	CG RAM [8]		/	?	O	_	o	←			ッ	ソ	マ	、	ö	■

Tabela 1.3.2. Sposób budowania nietypowych wzorców znakowych.

kody znaków w pamięci DD_RAM								adres w polu pamięci CG_RAM					wzorec znaku w polu danych CG_RAM																																		
7	6	5	4	3	2	1	0	5	4	3	2	1	0	7	6	5	4	3	2	1	0																										
0 0 0 0 x 0 0 0 (adres 0 lub 8)								0 0 0					0	0	0	x x x																															
													0	0	1	1	1	1	0	1 0 0 0 1																											
													0	1	0	1	0	0	1	1 0 0 0 1																											
													0	1	1	1	1	1	0	1 1 1 1 0																											
													1	0	0	1	0	0	0	1 0 1 0 0																											
													1	0	1	1	0	0	1	1 0 0 1 0																											
													1	1	0	0	0	0	1	1 0 0 0 1																											
													1	1	1	1	1	1	0	x x x 0 0 0 0 0 ← pozycja kursora																											
0 0 0 0 x 0 0 1 (adres 1 lub 9)								0 0 1					0	0	0	0 0 0 1 0																															
													0	0	1	1	0	0	0 0 1 0 0																												
													0	1	0	0	0	0	0	0 0 0 0 0																											
													0	1	1	1	1	0	0	0 1 1 1 0																											
													1	0	0	1	0	0	1	1 0 0 0 1																											
													1	0	1	1	0	0	1	1 0 0 0 1																											
													1	1	0	0	1	1	0	0 1 1 1 0																											
													1	1	1	1	1	1	0	0 0 0 0 0 ← pozycja kursora																											
0 0 0 0 x 1 1 1 (adres 7 lub 15)								1 1 1					0	0	0	[shaded]																															
													0	0	1									[shaded]																							
																													[shaded]															
													1	0	0																									[shaded]							
													1	0	1																																
1	1	0	[shaded]																																												
1	1	1									[shaded]																																				
wzór znaku 8																			[shaded]					[shaded]																							

1.3.2. System mikroprocesorowy FTSM_51.

Tabela 1.3.3. Opis wyprowadzeń złącza IDC40 systemu FTSM_51.

typ	opis	nazwa	numer		nazwa	opis	typ
I	przerwanie	INT0	1	2	INT1	przerwanie	I
I	sygnał potwierdzenia obecności I/O	ACK	3	4	IOCS	sygnał wywołania urządzenia I/O	O
O	sygnał odczytu	RD	5	6	WR	sygnał zapisu	O
I/O	magistrala danych	D0	7	8	A0	magistrala adresowa	O
I/O		D1	9	10	A1		O
I/O		D2	11	12	A2		O
I/O		D3	13	14	A3		O
I/O		D4	15	16	A4		O
I/O		D5	17	18	A5		O
I/O		D6	19	20	A6		O
I/O		D7	21	22	A7		O
I/O	końcówki μ C1	P1.7	23	24	P1.6	końcówki μ C1	I/O
I/O		P1.5	25	26	P1.4		I/O
I/O		P1.3	27	28	P1.2		I/O
I/O		P1.1	29	30	P1.0		I/O
I/O		P3.5	31	32	P3.4		I/O
O	linia dodatkowego kanału RS232	TDX	33	34	RDX	linia dodatkowego kanału RS232	I
O	zegar systemowy	CLK	35	36	PX0	linie	I/O
P	napięcie testowania	3V3	37	38	PX1	zarezerwowane	I/O
P	napięcie zasilania	VXX	39	40	GND	masa elektryczna	P

Uwagi:

- oznaczenia końcówek: I - wejście; O - wyjście; I/O - wejście/wyjście; P - zasilanie;
- w wersji 1.0.0, system FTSM nie posiada wewnętrznego kontrolera przerw - linie INT0 i INT1 są liniami własnymi mikrokontrolera AT89S8253;
- linie PX0 i PX1 są liniami mikrokontrolera μ C2 (kontrolera pomocniczego) i są zarezerwowane dla działań systemowych;
- wydajność prądowa wyjścia 3V3 (3,3V) jest ograniczona do 50 mA;
- wydajność prądowa wyjścia VXX (5,0V) jest ograniczona do 250 mA.

1.3.3. Mikrokontroler 80C51/52 - bity kontrolne.

Tabela 2.3.4. Wykaz bitów kontrolnych mikrokontrolerów 80C51/52 wg porządku alfabetycznego bitów kontrolnych.

nazwa bitu	adres bitu (hex)	numer bitu	nazwa bajtu	adres bajtu (hex)	funkcja	opis na stronie
AC	D6	D6	PSW	D0	flaga przeniesienia (BCD)	151
C/T	-	D2	TMOD	89	wybór funkcji układu licznika T0	37
C/T	-	D6	TMOD	89	wybór funkcji układu licznika T1	37
C/T2	C9	D1	T2CON	C8	wybór funkcji układu licznika T2	38
CP/RL2	C8	D0	T2CON	C8	bit wyboru trybu pracy T2	38
CY	D7	D7	PSW	D0	flaga przeniesienia	151
EA	AF	D7	IE	A8	bit aktywacji systemu przerwań	55
ES	AC	D4	IE	A8	bit aktywacji przerwań portu szer.	55
ET0	A9	D1	IE	A8	bit aktywacji przerwań licznika T0	55
ET1	AB	D3	IE	A8	bit aktywacji przerwań licznika T1	55
ET2	AD	D5	IE	A8	bit aktywacji przerwań licznika T2	55
EX0	A8	D0	IE	A8	bit aktywacji przerwań linii INT1	55
EX1	AA	D2	IE	A8	bit aktywacji przerwań linii INT1	55
EXEN2	CB	D3	T2CON	C8	bit aktywacji wejścia T2EX	38
EXF2	CE	D6	T2CON	C8	flaga detekcji sygnału T2EX	38
F0	D5	D5	PSW	D0	bit ogólnego przeznaczenia	151
GATE	-	D3	TMOD	89	bit uaktywnienia końcówki T0	37
GATE	-	D7	TMOD	89	bit uaktywnienia końcówki T1	37
GF0	-	D2	PCON	87	bit ogólnego przeznaczenia	51
GF1	-	D3	PCON	87	bit ogólnego przeznaczenia	51
IDL	-	D0	PCON	87	bit aktywacji stanu jałowego	51
IE0	89	D1	TCON	88	flaga zgłoszenia przerwania INT0	37
IE1	8B	D3	TCON	88	flaga zgłoszenia przerwania INT1	37
IT0	88	D0	TCON	88	flaga zgłoszenia przerwania T0	37
IT1	8A	D2	TCON	88	flaga zgłoszenia przerwania T1	37
M0	-	D0	TMOD	89	wybór trybu pracy układu T0	37
M0	-	D4	TMOD	89	wybór trybu pracy układu T1	37
M1	-	D1	TMOD	89	wybór trybu pracy układu T0	37
M1	-	D5	TMOD	89	wybór trybu pracy układu T1	37
OV	D2	D2	PSW	D0	flaga nadmiaru	151
P	D0	D0	PSW	D0	flaga parzystości	151

nazwa bitu	adres bitu (hex)	numer bitu	nazwa bajtu	adres bajtu (hex)	funkcja	opis na stronie
PD	-	D1	PCON	87	bit aktywacji obniż. poboru mocy	51
PS	BC	D4	IP	B8	bit priorytetu przerw. portu szereg.	55
PT0	B9	D1	IP	B8	bit priorytetu przerw. licznika T0	55
PT1	BB	D3	IP	B8	bit priorytetu przerw. licznika T1	55
PT2	BD	D5	IP	B8	bit priorytetu przerw. licznika T2	55
PX0	B8	D0	IP	B8	bit priorytetu przerw. linii INT0	55
PX1	BA	D2	IP	B8	bit priorytetu przerw. linii INT1	55
RB8	9A	D2	SCON	98	9 bit odbieranego znaku	50
RCLK	CD	D5	T2CON	C8	bit wyboru zegara odbiornika	38
REN	9C	D4	SCON	98	bit aktywacji odbioru portu szereg.	50
RI	98	D0	SCON	98	flaga odbioru bajtu	50
RS0	D3	D3	PSW	D0	wybór banku	151
RS1	D4	D4	PSW	D0	wybór banku	151
SM0	9F	D7	SCON	98	bit wyboru trybu pracy portu szer.	50
SM1	9E	D6	SCON	98	bit wyboru trybu pracy portu szer.	50
SM2	9D	D5	SCON	98	bit wyboru trybu pracy portu szer.	50
SMOD	-	D7	PCON	87	bit podwojenia szybk. transm. szer.	51
TB8	9B	D3	SCON	98	9 bit nadawanego znaku	50
TCLK	CC	D4	T2CON	C8	bit wyboru zegara nadajnika	38
TF0	8D	D5	TCON	88	flaga przepełnienia licznika T0	37
TF1	8F	D7	TCON	88	flaga przepełnienia licznika T1	37
TF2	CF	D7	T2CON	C8	flaga przepełnienia liczn. T2	38
TI	99	D1	SCON	98	flaga nadania bajtu	50
TR0	8C	D4	TCON	88	bit aktywacji zliczania przez T1	37
TR1	8E	D6	TCON	88	bit aktywacji zliczania przez T1	37
TR2	CA	D2	T2CON	C8	bit aktywacji zliczania przez T2	38

Tabela 1.3.5. Wykaz bitów kontrolnych mikrokontrolerów 80C51/52 oraz 89S8253 - wg porządku alfabetycznego rejestrów kontrolnych.

nazwa bitu	adres bitu (hex)	numer bitu	nazwa bajtu	adres bajtu (hex)	funkcja	opis na stronie
EX0	A8	D0	IE	A8	bit aktywacji przerwań linii INT1	55
ET0	A9	D1			bit aktywacji przerwań licznika T0	
EX1	AA	D2			bit aktywacji przerwań linii INT1	
ET1	AB	D3			bit aktywacji przerwań licznika T1	
ES	AC	D4			bit aktywacji przerwań portu szer.	
ET2	AD	D5			bit aktywacji przerwań licznika T2	
EA	AF	D7			bit aktywacji systemu przerwań	
PX0	B8	D0	IP	B8	bit priorytetu przerw. linii INT0	55
PT0	B9	D1			bit priorytetu przerw. licznika T0	
PX1	BA	D2			bit priorytetu przerw. linii INT1	
PT1	BB	D3			bit priorytetu przerw. licznika T1	
PS	BC	D4			bit priorytetu przerw. portu szereg.	
PT2	BD	D5			bit priorytetu przerw. licznika T2	
IDL	-	D0	PCON	87	bit aktywacji stanu jałowego	51
PD	-	D1			bit aktywacji obniż. poboru mocy	
GF0	-	D2			bit ogólnego przeznaczenia	
GF1	-	D3			bit ogólnego przeznaczenia	
SMOD	-	D7			bit podwojenia szybk. transm. szer.	
P	D0	D0	PSW	D0	flaga parzystości	151
OV	D2	D2			flaga nadmiaru	
RS0	D3	D3			wybór banku	
RS1	D4	D4			wybór banku	
F0	D5	D5			bit ogólnego przeznaczenia	
AC	D6	D6			flaga przeniesienia (BCD)	
CY	D7	D7			flaga przeniesienia	
RI	98	D0			SCON	
TI	99	D1	flaga nadania bajtu			
RB8	9A	D2	9 bit odbieranego znaku			
TB8	9B	D3	9 bit nadawanego znaku			
REN	9C	D4	bit aktywacji odbioru portu szereg.			
SM2	9D	D5	bit wyboru trybu pracy portu szer.			
SM1	9E	D6	bit wyboru trybu pracy portu szer.			
SM0	9F	D7	bit wyboru trybu pracy portu szer.			

nazwa bitu	adres bitu (hex)	numer bitu	nazwa bajtu	adres bajtu (hex)	funkcja	opis na stronie
CP/RL2	C8	D0	T2CON	C8	bit wyboru trybu pracy T2	38
C/T2	C9	D1			wybór funkcji układu licznika T2	
TR2	CA	D2			bit aktywacji zliczania przez T2	
EXEN2	CB	D3			bit aktywacji wejścia T2EX	
TCLK	CC	D4			bit wyboru zegara nadajnika	
RCLK	CD	D5			bit wyboru zegara odbiornika	
EXF2	CE	D6			flaga detekcji sygnału T2EX	
TF2	CF	D7			flaga przepełnienia liczn. T2	
IT0	88	D0	TCON	88	flaga zgłoszenia przerwania T0	37
IE0	89	D1			flaga zgłoszenia przerwania INTO	
IT1	8A	D2			flaga zgłoszenia przerwania T1	
IE1	8B	D3			flaga zgłoszenia przerwania INT1	
TR0	8C	D4			bit aktywacji zliczania przez T1	
TF0	8D	D5			flaga przepełnienia licznika T0	
TR1	8E	D6			bit aktywacji zliczania przez T1	
TF1	8F	D7			flaga przepełnienia licznika T1	
M0	-	D0	TMOD	89	wybór trybu pracy układu T0	37
M1	-	D1			wybór trybu pracy układu T0	
C/T	-	D2			wybór funkcji układu licznika T0	
GATE	-	D3			bit uaktywnienia końcówki T0	
M0	-	D4			wybór trybu pracy układu T1	
M1	-	D5			wybór trybu pracy układu T1	
C/T	-	D6			wybór funkcji układu licznika T1	
GATE	-	D7			bit uaktywnienia końcówki T1	

1.3.4. Mikrokontroler 80C51/52 - rejestry kontrolne.

nazwa rejestru: **IE**
 funkcja: rejestr aktywacji przerw - maska przerw
 adres: **A8h** (168)
 stan początkowy: 0x000000b

	IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
nazwy bitów:	EA	-	ET2	ES	ET1	EX1	ET0	EX0
adres bitów:	AFh	A Eh	ADh	ACH	ABh	AAh	A9h	A8h

IE.7 **EA** bit maskowania wszystkich przerw
 IE.6 - rezerwa
 IE.5 **ET2** maska przerywania od licznika T2
 IE.4 **ES** maska przerywania od portu transmisji szeregowej
 IE.3 **ET1** maska przerywania od licznika T1
 IE.2 **EX1** maska przerywania zewnętrznego od końcówki INT1
 IE.1 **ET0** maska przerywania od licznika T0
 IE.0 **EX0** maska przerywania zewnętrznego od końcówki INTO

znaczenie bitów i ich działanie:

IE.7 = 0 dezaktywuje cały system przerw
IE.7 = 1 aktywuje przerw. wskazane pozostałymi bitami maski
IE.x = 0 przerwanie zablokowane (x = 0 .. 5)
IE.x = 1 przerwanie aktywne (x = 0 .. 5)

nazwa rejestru: **IP**
 funkcja: rejestr priorytetu przerw
 adres: **B8h** (184)
 stan początkowy: xx000000b

	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
nazwy bitów:	-	-	PT2	PS	PT1	PX1	PT0	PX0
adres bitów:	BFh	BEh	BDh	BCh	BBh	BAh	B9h	B8h

IP.7 - rezerwa
 IP.6 - rezerwa
 IP.5 **PT2** priorytet przerywania od licznika T2
 IP.4 **PS** priorytet przerywania od portu transmisji szeregowej
 IP.3 **PT1** priorytet przerywania od licznika T1
 IP.2 **PX1** priorytet przerywania zewnętrznego od końcówki INT1
 IP.1 **PT0** priorytet przerywania od licznika T0
 IP.0 **PX0** priorytet przerywania zewnętrznego od końcówki INTO

znaczenie bitów i ich działanie:**IP.x = 0** przerwanie na poziomie podstawowym (x = 0 .. 5)**IP.x = 1** przerwanie na poziomie wysokim (x = 0 .. 5)

nazwa rejestru:

PCON

funkcja: rejestr statusu zasilania

adres: **87h** (135)

stan początkowy: 0xxx000b

nazwy bitów:

PCON.7	PCON.6	PCON.5	PCON.4	PCON.3	PCON.2	PCON.1	PCON.0
SMOD	-	-	-	GF1	GF0	PD	IDL

adres bitów:

- - - - - - - -

PCON.7 **SMOD** bit mnożnika szybkości transmisji szeregowej

PCON.6 - rezerwa

PCON.5 - rezerwa

PCON.4 - rezerwa

PCON.3 **GF1** bit ogólnego przeznaczenia (tylko układy CHMOS)PCON.2 **GF0** bit ogólnego przeznaczenia (tylko układy CHMOS)PCON.1 **PD** bit aktywacji obniżonego poboru mocyPCON.0 **IDL** bit aktywacji pracy w trybie jałowymznaczenie bitów i ich działanie:

SMOD = 0 stan podstawowy: częstotliwość taktowania portu transmisji szeregowej w trybie 1 i 2 wyznacza licznik T1; w trybie 2 częstotliwość taktowania wynosi:
 $f = f_{osc}/4$

SMOD = 1 szybkość transmisji jest podwojona w stosunku do stanu podstawowego

GF0, GF1 bity zmieniane wyłącznie programowo do dowolnego wykorzystania

PD = 0 praca normalna

PD = 1 praca z obniżonym poborem mocy (patrz uwagi na str.12)

IDL = 0 praca normalna

IDL = 1 praca w trybie jałowym (patrz uwagi na str.12)

nazwa rejestru: **PSW**
 funkcja: rejestr statusu programu
 adres: **D0h** (208)
 stan początkowy: 00000000b

	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
nazwy bitów:	CY	AC	F0	RS1	RS0	OV	F1	P
adres bitów:	D7h	D6h	D5h	D4h	D3h	D2h	D1h	D0h

PSW.7 **CY** flaga przeniesienia
 PSW.6 **AC** flaga przeniesienia pomocniczego
 PSW.5 **F0** bit ogólnego przeznaczenia
 PSW.4 **RS1** bit deklaracji numeru banku rejestrów roboczych
 PSW.3 **RS0** bit deklaracji numeru banku rejestrów roboczych
 PSW.2 **OV** flaga stanu przepełnienia
 PSW.1 **F1** bit ogólnego przeznaczenia
 PSW.0 **P** flaga parzystości danej w akumulatorze

znaczenie bitów i ich działanie:

CY bit zmieniany sprzętowo podczas wykonywania operacji arytmetycznych - sygnalizuje stan przeniesienia lub pożyczki; w rozkazach typu bitowego jest oznaczony literą **C**

AC bit zmieniany sprzętowo podczas wykonywania operacji arytmetycznych - sygnalizuje stan przeniesienia lub pożyczki z bitu 3; wykorzystywany przez rozkaz korekcji dziesiętnej dodawania w kodzie BCD (rozkaz **DAA**)

F0, F1 bity zmieniane wyłącznie programowo do dowolnego wykorzystania

RS1, RS0 bity zmieniane wyłącznie programowo - określają bieżący numer banku rejestrów roboczych **R0..R7**

RS1	RS0	nr banku	adres pola RAM
0	0	0	00h - 07h
0	1	1	08h - 0Fh
1	0	2	10h - 17h
1	1	3	18h - 1Fh

OV bit zmieniany sprzętowo podczas wykonywania operacji arytmetycznych - sygnalizuje stan przekroczenia zakresu dla liczb w kodzie U2

P bit zmieniany sprzętowo w każdym cyklu maszynowym - określa parzystość danej w akumulatorze: gdy P=1 to akumulator zawiera parzystą liczbę jedynek

nazwa rejestru: **SCON**
 funkcja: rejestr sterowania portem transmisji szeregowej
 adres: **98h** (152)
 stan początkowy: 00000000b

	SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
nazwy bitów:	EA	-	ET2	ES	ET1	EX1	ET0	EX0
adres bitów:	AFh	A Eh	ADh	A Ch	ABh	AAh	A9h	A8h

SCON.7 **SM0** bit wyboru trybu pracy
 SCON.6 **SM1** bit wyboru trybu pracy
 SCON.5 **SM2** bit wyboru trybu pracy
 SCON.4 **REN** aktywacja odbioru
 SCON.3 **TB8** 9 bit nadawanego bajtu
 SCON.2 **RB8** 9 bit odbieranego bajtu
 SCON.1 **TI** flaga wysłania bajtu
 SCON.0 **RI** flaga odebrania bajtu

znaczenie bitów i ich działanie:

SM0, SM1 bity zmieniane programowo - określają tryb pracy portu transmisji szeregowej (patrz tabela 1.2.10)
SM2 bit zmieniany programowo; gdy SM2=1 to aktywowana jest komunikacja wieloprocesorowa

SM0	SM1	tyb	opis	szybkość transmisji
0	1	0	transmisja synchroniczna, słowo 8-bitowe	$f_{osc}/12$
0	1	1	transmisja asynchroniczna, słowo 8-bitowe	definiowana przez licznik T1
1	0	2	transmisja asynchroniczna, słowo 9-bitowe	$f_{osc}/64$ lub $f_{osc}/32$
1	1	3	transmisja asynchroniczna słowo 9-bitowe	definiowana przez licznik T1

SM0 = 0 dezaktywuje cały system przerwania
IE.7 = 1 aktywuje przerwania wskazane pozostałymi bitami maski
IE.x = 0 przerwanie zablokowane (x = 0 .. 5)
IE.x = 1 przerwanie aktywne

nazwa rejestru: **TCON**
 funkcja: rejestr statusu liczników T0 i T1
 adres: **88h** (136)
 stan początkowy: 00000000b

	TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
nazwy bitów:	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
adres bitów:	8Fh	8Eh	8Dh	8Ch	8Bh	8Ah	89h	88h

TCON.7 **TF1** flaga przepełnienia licznika T1
 TCON.6 **TR1** bit włączania licznika T1
 TCON.5 **TF0** flaga przepełnienia licznika T0
 TCON.4 **TR0** bit włączania licznika T0
 TCON.3 **IE1** flaga zgłoszenia przerwania zewnętrznego INT1
 TCON.2 **IT1** bit ustawiania sposobu przyjmowania przerwania INT1
 TCON.1 **IE0** flaga zgłoszenia przerwania zewnętrznego INT0
 TCON.0 **IT0** bit ustawiania sposobu przyjmowania przerwania INT0

znaczenie bitów i ich działanie:

- TF1** flaga układu przerwań; bit ustawiany sprzętowo po przepełnieniu licznika T1; po przyjęciu przerwania jest automatycznie kasowany
- TR1** zmieniany programowo; służy do włączania (TR1=1) lub wyłączenia (TR1=0) licznika T1
- TF0** flaga układu przerwań; bit ustawiany sprzętowo po przepełnieniu licznika T0; po przyjęciu przerwania jest automatycznie kasowany
- TR0** zmieniany programowo; służy do włączania (TR0=1) lub wyłączenia (TR0=0) licznika T0
- IE1** flaga układu przerwań; stan bitu jest negacją stanu końcówki INT0 lub jest on ustawiany (IE1=1) po wykryciu opadającego zbocza sygnału INT1
- IT1** zmieniany programowo; stan bitu ustala sposób zgłaszania przerwania od końcówki INT1: gdy IT1=0 to zgłaszanie poziomem niskim; gdy IT1=1 to zgłaszanie opadającym zboczem sygnału przerwania
- IE0** flaga układu przerwań; stan bitu jest negacją stanu końcówki INT0 lub jest on ustawiany (IE0=1) po wykryciu opadającego zbocza sygnału INT0
- IT0** zmieniany programowo; stan bitu ustala sposób zgłaszania przerwania od końcówki INT1: gdy IT1=0 to zgłaszanie poziomem niskim; gdy IT1=1 to zgłaszanie opadającym zboczem sygnału przerwania

nazwa rejestru: **TMOD**
 funkcja: rejestr nastaw trybu pracy liczników T0 i T1
 adres: **89h** (137)
 stan początkowy: 00000000b

	TMOD.7	TMOD.6	TMOD.5	TMOD.4	TMOD.3	TMOD.2	TMOD.1	TMOD.0
nazwy bitów:	GATE	C/T	M1	M0	GATE	C/T	M1	M0
	licznik T1				licznik T0			

TMOD.7 **GATE** bit aktywacji bramkowania zewnętrznego licznika T1
 TMOD.6 **C/T** bit wyboru funkcji licznika T1
 TMOD.5 **M1** bit wyboru pracy licznika T1
 TMOD.4 **M0** bit wyboru pracy licznika T1
 TMOD.3 **GATE** bit aktywacji bramkowania zewnętrznego licznika T0
 TMOD.2 **C/T** bit wyboru funkcji licznika T0
 TMOD.1 **M1** bit wyboru pracy licznika T0
 TMOD.0 **M0** bit wyboru pracy licznika T0

znaczenie bitów i ich działanie:

GATE gdy $GATE_{(0,1)}=1$ oraz $TR_{(0,1)}=1$ (w TCON) to licznik $T_{(0,1)}$ pracuje wyłącznie gdy stan końcówki $INT_{(0,1)}=1$ (sterowanie sprzętowe pracą licznika);
 gdy $GATE_{(0,1)}=0$ to licznik $T_{(0,1)}$ pracuje gdy $TR_{(0,1)}=1$. (sterowanie programowe)

C/T służy do określenia trybu pracy licznika: gdy $C/T=0$ to licznik jest taktowany sygnałem wewnętrznym ($f_{OSC}/12$) i jest czasomierzem; gdy $C/T=1$ to licznik zlicza impulsy z końcówek T0 lub (i) T1.

M1, M0 bity określają sposób pracy liczników T1 i T2

M1	M0	tryb	opis
0	0	0	licznik 13-bitowy
0	1	1	licznik 16-bitowy
1	0	2	licznik 8-bitowy z przeładowaniem
1	1	3	2 liczniki 8 bitowe

w trybie 3, licznik TL0 jest sterowany sygnałami sterowania standardowego licznika T0; licznik TL1 jest sterowany bitem TR1; rejestry TL1 i TH1 licznika T1 nie są używane

nazwa rejestru: **T2CON**
 funkcja: rejestr sterowania licznikiem T2 (80C52, 89S8253)
 adres: **C8h** (200)
 stan początkowy: 00000000b

	T2CON.7	T2CON.6	T2CON.5	T2CON.4	T2CON.3	T2CON.2	T2CON.1	T2CON.0
nazwy bitów:	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
adres bitów:	CFh	CEh	CDh	CCh	CBh	CAh	C9h	C8h

T2CON.7 **TF2** flaga przepełnienia licznika T2
 T2CON.6 **EXF2** flaga stanu aktywnego sygnału z końcówki T2EX (P1.1)
 T2CON.5 **RCLK** włącznik sygnału taktowania nadajnika UART
 T2CON.4 **TCLK** włącznik sygnału taktowania nadajnika UART
 T2CON.3 **EXEN2** aktywuje wejście T2EX
 T2CON.2 **TR2** włącznik licznika T2
 T2CON.1 **C/T2** selektor źródła taktowania licznika T2
 T2CON.0 **CP/RL2** selektor trybu pracy licznika T2

znaczenie bitów i ich działanie:

TF2 flaga układu przerwań; bit ustawiany sprzętowo po przepełnieniu licznika T2 - gdy licznik nie jest używany do taktowania elementów portu UART (**RCLK=0** oraz **TCLK=0**), bit musi być kasowany programowo

EXF2 flaga układu przerwań; bit ustawiany sprzętowo przez opadające zbocze sygnału zewnętrznego z końcówki T2EX; bit musi być kasowany programowo

RCLK zmieniany programowo; gdy **RCLK=1** to odbiornik portu UART, który pracuje w trybie 1 lub 3, jest taktowany impulsami przepełnienia licznika T2; gdy **RCLK=0** to odbiornik portu UART jest taktowany impulsami przepełnienia licznika T1

TCLK zmieniany programowo; gdy **TCLK=1** to nadajnik portu UART, który pracuje w trybie 1 lub 3, jest taktowany impulsami przepełnienia licznika T2; gdy **TCLK=0** to nadajnik portu UART jest taktowany impulsami przepełnienia licznika T1

EXEN2 zmieniany programowo; gdy licznik nie jest używany do taktowania elementów portu UART (**RCLK=0** oraz **TCLK=0**) oraz gdy **EXEN2=1** to opadające zbocze sygnału zewnętrznego, wprowadzonego przez końcówkę T2EX, powoduje przechwycenie stanu licznika lub jego przeładowanie zawartością rejestru RCAP; gdy **EXEN2=1** to licznik 2 ignoruje zdarzenia z końcówki T2EX

- TR2** zmieniany programowo; stan bitu decyduje o dołączeniu (TR2=1) lub odłączeniu sygnału taktowania licznika
- C/T2** zmieniany programowo; służy do określenia trybu pracy licznika: gdy C/T=0 to licznik jest taktowany sygnałem wewnętrznym ($f_{OSC}/12$) i jest czasomierzem; gdy C/T=1 to licznik zlicza impulsy z końcówki T2 (P1.0)
- CP/RL2** zmieniany programowo; ustawia tryb pracy licznika gdy RCLK=0 i TCLK=0: gdy CP/RL2=0 to licznik pracuje w trybie automatycznego przeładowania; gdy CP/RL2=1 to licznik pracuje w try-bie przechwytywania.

1.3.5. Mikrokontroler 89S8253.

Tabela 1.3.6. Pole SFR mikrokontrolera 89S8253

0F8h									0FFh
0F0h	B 00000000								0F7h
0E8h									0EFh
0E0h	ACC 00000000								0E7h
0D8h									0DFh
0D0h	PSW 00000000					SPCR 00000100			0D7h
0C8h	T2CON 00000000	T2MOD xxxxxx00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFh
0C0h									0C7h
0B8h	IP xx000000	SADEN 00000000							0BFh
0B0h	P3 11111111							IPH xx000000	0B7h
0A8h	IE 0x000000	SADR 00000000	SPSR 000xxx00						0AFh
0A0h	P2 11111111						WDTRST (tylko zapis)	WDTCON 00000000	0A7h
098h	SCON 00000000	SBUF xxxxxxx							09Fh
090h	P1 11111111						EECON xx000011		097h
088h	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR xxxxxxx0	CLKREG xxxxxxx0	08Fh
080h	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000	SPDR #####	PCON 0xxx0000	087h

Uwaga: symbol # oznacza, że przyjmowana jest wartość 0 po tzw. "zimnym kasowaniu" oraz wartość pozostaje bez zmian po "gorącym kasowaniu".

Tabela 1.3.7. Wykaz dodatkowych bitów kontrolnych mikrokontrolera 89S8253, które omówiono w skrypcie (wg porządku alfabetycznego bitów).

nazwa bitu	adres bitu (hex)	numer bitu	nazwa bajtu	adres bajtu (hex)	funkcja	opis na stronie
DCEN	-	D0	T2MOD	C9	bit wyboru trybu pracy T2	39
DISRTO	-	D3	WDTCN	A7	bit wyboru trybu pracy końcówki RST	60
DPS	-	D2	EECON	96	bit wyboru wskaźnika DPTR	58
HWDT	-	D2	WDTCN	A7	bit ustaw. trybu ster. pracą WDT	60
PS0	-	D5	WDTCN	A7	bit wyboru pojemności licz. WDT	60
PS1	-	D6	WDTCN	A7	bit wyboru pojemności licz. WDT	60
PS2	-	D7	WDTCN	A7	bit wyboru pojemności licz. WDT	60
T2OE	-	D1	T2MOD	C9	bit wyboru trybu pracy T2	39
WDIDLE	-	D4	WDTCN	A7	bit ust. trybu WDT w stanie jałowym	60
WDTEN	-	D0	WDTCN	A7	bit aktywowania układu WDT	60
WSWRST	-	D1	WDTCN	A7	bit kasowania układu WDT	60

Tabela 1.3.8. Wykaz dodatkowych bitów kontrolnych mikrokontrolera 89S8253, które omówiono w skrypcie (wg porządku alfabetycznego rejestrów kontrolnych).

nazwa bitu	adres bitu (hex)	numer bitu	nazwa bajtu	adres bajtu (hex)	funkcja	opis na stronie
DPS	-	D2	EECON	96	bit wyboru wskaźnika DPTR	58
DCEN	-	D0	T2MOD	C9	bit wyboru trybu pracy T2	39
T2OE	-	D1			bit wyboru trybu pracy T2	
WDTEN	-	D0	WDTCN	A7	bit aktywowania układu WDT	60
WSWRST	-	D1			bit kasowania układu WDT	
HWDT	-	D2			bit ustaw. trybu ster. pracą WDT	
DISRTO	-	D3			bit wyboru trybu pracy końcówki RST	
WDIDLE	-	D4			bit ust. trybu WDT w stanie jałowym	
PS0	-	D5			bit wyboru pojemności licz. WDT	
PS1	-	D6			bit wyboru pojemności licz. WDT	
PS2	-	D7			bit wyboru pojemności licz. WDT	

1.4. Skrócona lista rozkazów.

oznaczenia rejestrów:

- A, ACC - akumulator
- DPTR - wskaźnik danych
- B - rejestr B
- PC - licznik rozkazów
- Ri - rejestr roboczy R0, R1
- Rn - rejestr roboczy R0..R7
- SP - wskaźnik stosu

oznaczenia bitów:

- AC - flaga przeniesienia pomocniczego
- C, CY - flaga przeniesienia
- OV - flaga nadmiaru

oznaczenia adresów:

- adr - adres 8 bitowy w polu wewnętrznej pamięci RAM i SFR (także: adr1, adr2)
- adr_11 - adres 11 bitowy w polu pamięci programu
- adr_16 - adres 16 bitowy w polu pamięci programu
- bit - adres bitu w polu bitowym RAM lub SFR
- d - przesunięcie adresu w polu pamięci programu, < -128, 127>

oznaczenia argumentów:

- n - argument 8 bitowy
- nn - argument 16 bitowy

inne oznaczenia:

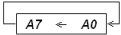
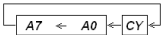

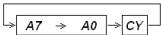
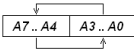
- @ - oznaczenie adresowania pośredniego
- # - oznaczenie argumentu bezpośredniego
- X - oznacza zawartość rejestru X
- (X) - oznacza zawartość pamięci o adresie X
- [X] - oznacza skrócony opis działania operacji lub działanie warunkowe - dokładne omówienie podane jest w pełnym opisie instrukcji.

Tabela 2.4.1. Wykaz instrukcji zmieniających stan flag CY, AC i OV.

rozkaz	znaczniki			rozkaz	znaczniki			rozkaz	znaczniki		
	C	AC	OV		C	AC	OV		C	AC	OV
ADD	x	x	x	CPL C	x			ORL C,/bit	x		
ADDC	x	x	x	DA	x			RLC	x		
ANL C,bit	x			DIV	0		x	RRC	x		
ANL C,/bit	x			MOV C,bit	x			SETB C	1		
CJNE	x			MUL	0		x	SUBB	x	x	x
CLR C	x			ORL C,bit	x						

Tabela 1.4.2. Wykaz instrukcji - zestawienie wg funkcji.

mnemonik	operacja	cykle	uwagi
przesyłanie bajtów ..			
MOV A,adr	$A \leftarrow (\text{adr})$	1	
MOV A,#n	$A \leftarrow n$	1	
MOV A,Rn	$A \leftarrow Rn$	1	
MOV A,@Ri	$A \leftarrow (Ri)$	1	
MOV adr,A	$(\text{adr}) \leftarrow A$	1	
MOV adr,#n	$(\text{adr}) \leftarrow n$	2	
MOV adr,Rn	$(\text{adr}) \leftarrow Rn$	2	
MOV adr,@Ri	$(\text{adr}) \leftarrow (Ri)$	2	
MOV adr1,adr2	$(\text{adr}1) \leftarrow (\text{adr}2)$	2	
MOV DPTR,#nn	$DPTR \leftarrow nn$	2	
MOV Rn,A	$Rn \leftarrow A$	1	
MOV Rn,adr	$Rn \leftarrow (\text{adr})$	2	
MOV Rn,#n	$Rn \leftarrow n$	1	
MOV @Ri,A	$(Ri) \leftarrow A$	1	
MOV @Ri,adr	$(Ri) \leftarrow (\text{adr})$	2	
MOV @Ri,#n	$(Ri) \leftarrow n$	1	
XCH A,adr	$A \leftrightarrow (\text{adr})$	1	
XCH A,Rn	$A \leftrightarrow Rn$	1	
XCH A,@Ri	$A \leftrightarrow (Ri)$	1	
XCHD A,@Ri	$A_{3,0} \leftrightarrow (Ri)_{3,0}$	1	
przesyłanie bajtów w trybie pracy mikroprocesorowej ..			
MOVX A,@Ri	$A \leftarrow (Ri)$	2	} adres 8 bitowy w polu pamięci danych
MOVX @Ri,A	$(Ri) \leftarrow A$	2	
MOVX A,@DPTR	$A \leftarrow (DPTR)$	2	} adres 16 bitowy w polu pamięci danych
MOVX @DPTR,A	$(DPTR) \leftarrow A$	2	
MOVC A,@A+DPTR	$A \leftarrow (A + DPTR)$	2	} adres 16 bitowy w polu pamięci programu
MOVC A,@A+PC	$A \leftarrow (A + PC)$	2	
operacje arytmetyczne ..			
ADD A,adr	$A \leftarrow A + (\text{adr})$	1	
ADD A,#n	$A \leftarrow A + n$	1	
ADD A,Rn	$A \leftarrow A + Rn$	1	
ADD A,@Ri	$A \leftarrow A + (Ri)$	1	
ADDC A,adr	$A \leftarrow A + (\text{adr}) + CY$	1	
ADDC A,#n	$A \leftarrow A + n + CY$	1	
ADDC A,Rn	$A \leftarrow A + Rn + CY$	1	
ADDC A,@Ri	$A \leftarrow A + (Ri) + CY$	1	
SUBB A,adr	$A \leftarrow A - (\text{adr}) - CY$	1	
SUBB A,#n	$A \leftarrow A - n - CY$	1	
SUBB A,Rn	$A \leftarrow A - Rn - CY$	1	
SUBB A,@Ri	$A \leftarrow A - (Ri) - CY$	1	

mnemonik	operacja	cykle	uwagi
operacje arytmetyczne (cd) ..			
INC A	$A \leftarrow A + 1$	1	
INC adr	$(adr) \leftarrow (adr) + 1$	1	
INC DPTR	$DPTR \leftarrow DPTR + 1$	2	
INC Rn	$Rn \leftarrow Rn + 1$	1	
INC @Ri	$(Ri) \leftarrow (Ri) + 1$	1	
DEC A	$A \leftarrow A - 1$	1	
DEC adr	$(adr) \leftarrow (adr) - 1$	1	
DEC Rn	$Rn \leftarrow Rn - 1$	1	
DEC @Ri	$(Ri) \leftarrow (Ri) - 1$	1	
DA A		1	
DIV AB	$A, B \leftarrow A / B$	4	
MUL AB	$B, A \leftarrow A * B$	4	
operacje logiczne ..			
ANL A,adr	$A \leftarrow A \wedge (adr)$	1	
ANL A,#n	$A \leftarrow A \wedge n$	1	
ANL A,Rn	$A \leftarrow A \wedge Rn$	1	
ANL A,@Ri	$A \leftarrow A \wedge (Ri)$	1	
ANL adr,A	$(adr) \leftarrow (adr) \wedge A$	1	
ANL adr,#n	$(adr) \leftarrow (adr) \wedge n$	2	
CLR A	$A \leftarrow 0$	1	
CPL A	$A \leftarrow /A$	1	
ORL A,adr	$A \leftarrow A \vee (adr)$	1	
ORL A,#n	$A \leftarrow A \vee n$	1	
ORL A,Rn	$A \leftarrow A \vee Rn$	1	
ORL A,@Ri	$A \leftarrow A \vee (Ri)$	1	
ORL adr,A	$(adr) \leftarrow (adr) \vee A$	1	
ORL adr,#n	$(adr) \leftarrow (adr) \vee n$	2	
XRL A,adr	$A \leftarrow A \oplus (adr)$	1	
XRL A,#n	$A \leftarrow A \oplus n$	1	
XRL A,Rn	$A \leftarrow A \oplus Rn$	1	
XRL A,@Ri	$A \leftarrow A \oplus (Ri)$	1	
XRL adr,A	$(adr) \leftarrow (adr) \oplus A$	1	
XRL adr,#n	$(adr) \leftarrow (adr) \oplus n$	2	
RL A		1	
RLC A		1	
RR A		1	
RRC A		1	
SWAP A		1	

mnemonik	operacja	cykle	uwagi
operacje na bitach ..			
CLR bit	$(\text{bit}) \leftarrow 0$	1	
CLR C	$CY \leftarrow 0$	1	
CPL bit	$(\text{bit}) \leftarrow \neg(\text{bit})$	1	
CPL C	$CY \leftarrow \neg CY$	1	
ANL C,bit	$CY \leftarrow CY \wedge (\text{bit})$	2	
ANL C,/bit	$CY \leftarrow CY \wedge (\text{bit})$	2	
JC d	$[PC \leftarrow PC + d]$	2	skok gdy $CY = 1$
JNC d	$[PC \leftarrow PC + d]$	2	skok gdy $CY = 0$
JB bit,d	$[PC \leftarrow PC + d]$	2	skok gdy $\text{bit} = 1$
JNB bit,d	$[PC \leftarrow PC + d]$	2	skok gdy $\text{bit} = 0$
JBC bit,d	$[PC \leftarrow PC + d]$	2	skok gdy $\text{bit} = 1$; $(\text{bit}) \leftarrow 0$
MOV bit,C	$(\text{bit}) \leftarrow CY$	2	
MOV C,bit	$CY \leftarrow (\text{bit})$	1	
ORL C,bit	$CY \leftarrow CY \vee (\text{bit})$	2	
ORL C,/bit	$CY \leftarrow CY \vee (\text{bit})$	1	
SETB bit	$(\text{bit}) \leftarrow 1$	1	
SETB C	$CY \leftarrow 1$	1	
operacje rozgałęzień ..			
AJMP adr_11	$PC \leftarrow \text{adr_11}$	2	skok w obrębie strony
LJMP adr_16	$PC \leftarrow \text{adr_16}$	2	
SJMP d	$PC \leftarrow PC + d$	2	
JMP @A + DPTR	$PC \leftarrow A + \text{DPTR}$	2	
JZ d	$[PC \leftarrow PC + d]$	2	skok gdy $A = 0$
JNZ d	$[PC \leftarrow PC + d]$	2	skok gdy $A \neq 0$
CJNE A,adr,d	$[PC \leftarrow PC + d]$	2	skok gdy $A \neq (\text{adr})$
CJNE A,#n,d	$[PC \leftarrow PC + d]$	2	skok gdy $A \neq n$
CJNE Rn,#n,d	$[PC \leftarrow PC + d]$	2	skok gdy $Rn \neq n$
CJNE @Ri, #n,d	$[PC \leftarrow PC + d]$	2	skok gdy $(Ri) \neq n$
DJNZ Rn,d	$Rn \leftarrow Rn - 1$ $[PC \leftarrow PC + d]$	2	skok gdy $Rn \neq 0$
DJNZ adr,d	$(\text{adr}) \leftarrow (\text{adr}) - 1$ $[PC \leftarrow PC + d]$	2	skok gdy $(\text{adr}) \neq 0$
NOP		1	
ACALL adr_11	$[(SP) \leftarrow PC]$ $[SP \leftarrow SP + 2]$ $PC \leftarrow \text{adr_11}$	2	automatyczny zapis na stos stanu licznika PC - wywołanie w obrębie strony pamięci programu
LCALL adr_16	$[(SP) \leftarrow PC]$ $[SP \leftarrow SP + 2]$ $PC \leftarrow \text{adr_16}$	2	automatyczny zapis na stos stanu licznika PC
RET	$PC \leftarrow (SP)$ $[SP \leftarrow SP - 2]$	2	automatyczny odczyt ze stosu stanu licznika PC - powrót z podprogramu

mnemonik	operacja	cykle	uwagi
operacje rozgałęzień (cd) ..			
RETI	PC ← (SP) [SP ← SP - 2]	2	automatyczny odczyt ze stosu stanu licznika PC (powrót z przerwania)
PUSH adr	SP ← SP + 1 (SP) ← (adr)	2	
POP adr	(adr) ← (SP) SP ← SP - 1	2	

Tabela 1.4.3. Wykaz instrukcji - zestawienie wg kolejności numerów kodu.

00 NOP	20 JB bit,d	40 JC d	60 JZ d
01 AJMP adr11	21 AJMP adr11	41 AJMP adr11	61 AJMP adr11
02 LJMP adr16	22 RET	42 ORL ad,A	62 XRL ad,A
03 RRA	23 RL A	43 ORL ad,#n	63 XRL ad,#n
04 INC A	24 ADD A,#n	44 ORL A,#n	64 XRLA,#n
05 INC ad	25 ADD A,ad	45 ORL A,ad	65 XRL A,ad
06 INC @R0	26 ADD A,@R0	46 ORL A,@R0	66 XRL A,@R0
07 INC @R1	27 ADD A,@R1	47 ORL A,@R1	67 XRL A,@R1
08 INC R0	28 ADD A,R0	48 ORL A,R0	68 XRL A,R0
09 INC R1	29 ADD A,R1	49 ORL A,R1	69 XRL A,R1
0A INC R2	2A ADD A,R2	4A ORL A,R2	6A XRL A,R2
0B INC R3	2B ADD A,R3	4B ORL A,R3	6B XRL A,R3
0C INC R4	2C ADD A,R4	4C ORL A,R4	6C XRL A,R4
0D INC R5	2D ADD A,R5	4D ORL A,R5	6D XRL A,R5
0E INC R6	2E ADD A,R6	4E ORL A,R6	6E XRL A,R6
0F INC R7	2F ADD A,R7	4F ORL A,R7	6F XRL A,R7
10 JBC bit,d	30 JNB bit,d	50 JNC d	70 JNZ d
11 ACALL adr11	31 ACALL adr11	51 ACALL adr11	71 ACALL adr11
12 ACALL adr16	32 RETI	52 ANL ad,A	72 ORL C,bit
13 RRC A	33 RLC A	53 ANL ad,#n	73 JMP @A+DPTR
14 DEC A	34 ADDC A,#n	54 ANL A,#n	74 MOV A,#n
15 DEC ad	35 ADDC A,ad	55 ANL A,ad	75 MOV ad,#n
16 DEC @R0	36 ADDC A,@R0	56 ANL A,@R0	76 MOV @R0,#n
17 DEC @R1	37 ADDC A,@R1	57 ANL A,@R1	77 MOV @R1,#n
18 DEC R0	38 ADDC A,R0	58 ANL A,R0	78 MOV R0,#n
19 DEC R1	39 ADDC A,R1	59 ANL A,R1	79 MOV R1,#n
1A DEC R2	3A ADDC A,R2	5A ANL A,R2	7A MOV R2,#n
1B DEC R3	3B ADDC A,R3	5B ANL A,R3	7B MOV R3,#n
1C DEC R4	3C ADDC A,R4	5C ANL A,R4	7C MOV R4,#n
1D DEC R5	3D ADDC A,R5	5D ANL A,R5	7D MOV R5,#n
1E DEC R6	3E ADDC A,R6	5E ANL A,R6	7E MOV R6,#n
1F DEC R7	3F ADDC A,R7	5F ANL A,R7	7F MOV R7,#n

Wykaz instrukcji - zestawienie wg kolejności numerów kodu (cd).

80 SJMP d	A0 ORL C,/bit	C0 PUSH ad	E0 MOVX A,@DPTR
81 AJMP adr11	A1 AJMP adr11	C1 AJMP adr11	E1 AJMP adr11
82 ANL C,bit	A2 MOV C,bit	C2 CLR ad	E2 MOVX A,@R0
83 MOVC A,@A+PC	A3 INC DPTR	C3 CLR C	E3 MOVX A,@R1
84 DIV AB	A4 MUL AB	C4 SWAP A	E4 CLR A
85 MOV ad,ad	A5 -	C5 XCH A,ad	E5 MOV A,ad
86 MOV ad,@R0	A6 MOV @R0,ad	C6 XCH A,@R0	E6 MOV A,@R0
87 MOV ad,@R1	A7 MOV @R1,ad	C7 XCH A,@R1	E7 MOV A,@R1
88 MOV ad, R0	A8 MOV R0,ad	C7 XCH A,R0	E8 MOV A,R0
89 MOV ad, R1	A9 MOV R1,ad	C9 XCH A,R1	E9 MOV A,R1
8A MOV ad, R2	AA MOV R2,ad	CA XCH A,R2	EA MOV A,R2
8B MOV ad, R3	AB MOV R3,ad	CB XCH A,R3	EB MOV A,R3
8C MOV ad, R4	AC MOV R4,ad	CC XCH A,R4	EC MOV A,R4
8D MOV ad, R5	AD MOV R5,ad	CD XCH A,R5	ED MOV A,R5
8E MOV ad, R6	AE MOV R6,ad	CE XCH A,R6	EE MOV A,R6
8F MOV ad, R7	AF MOV R7,ad	CF XCH A,R7	EF MOV A,R7
90 MOV DPTR,#nn	B0 ANL C,/bit	D0 POP ad	F0 MOVX @DPTR,A
91 ACALL adr11	B1 ACALL adr11	D1 ACALL adr11	F1 ACALL adr11
92 MOV bit,C	B2 CPL bit	D2 SETB bit	F2 MOVX @R0,A
93 MOVCA,@A+DPTR	B3 CPL C	D3 SETB C	F3 MOVX @R1,A
94 SUBB A,#n	B4 CJNE A,#n,d	D4 DA A	F4 CPL A
95 SUBB A,ad	B5 CJNE A,ad,d	D5 DJNZ ad,d	F5 MOV ad,A
96 SUBB A,@R0	B6 CJNE @R0,#n,d	D6 XCHD A,@R0	F6 MOV @R0,A
97 SUBB A,@R1	B7 CJNE @R1,#n,d	D7 XCHD A,@R1	F7 MOV @R1,A
98 SUBB A,R0	B8 CJNE R0,#n,d	D8 DJNZ R0,d	F8 MOV R0,A
99 SUBB A,R1	B9 CJNE R1,#n,d	D9 DJNZ R1,d	F9 MOV R1,A
9A SUBB A,R2	BA CJNE R2,#n,d	DA DJNZ R2,d	FA MOV R2,A
9B SUBB A,R3	BB CJNE R3,#n,d	DB DJNZ R3,d	FB MOV R3,A
9C SUBB A,R4	BC CJNE R4,#n,d	DC DJNZ R4,d	FC MOV R4,A
9D SUBB A,R5	BD CJNE R5,#n,d	DD DJNZ R5,d	FD MOV R5,A
9E SUBB A,R6	BE CJNE R6,#n,d	DE DJNZ R6,d	FE MOV R6,A
9F SUBB A,R7	BF CJNE R7,#n,d	DF DJNZ R7,d	FF MOV R7,A

1.5. Pełna lista rozkazów.

W rozdziale opisano wszystkie instrukcje dla mikrokontrolerów z rodziny MCS-51. Oznaczenia rejestrów, bitów, adresów i argumentów są takie same jak w poprzednim rozdziale. Ponadto, do bardziej ogólnego opisu instrukcji użyto następujących oznaczeń:

- <dst> - adres argumentu instrukcji i miejsce zapisu wyniku operacji
- <dst_bit> - adres argumentu bitowego instrukcji i miejsce zapisu wyniku operacji
- <reg_16> - stan rejestru DPTR lub PC
- <src> - adres dodatkowego argumentu instrukcji
- <src_bit> - adres dodatkowego argumentu bitowego instrukcji

ACALL adr_11

wywoływanie podprogramu
na stronie pamięci programu
(ang. absolute subroutine call)

opis: Instrukcja bezwarunkowo wywołuje podprogram definiowany przez 11-bitowy adres bezpośredni.

bajty/cykle: 2/2

kodowanie:

a ₁₀	a ₉	a ₈	1	0	0	0	1	a ₇	a ₆	a ₅	a ₄	a ₃	a ₂	a ₁	a ₀
-----------------	----------------	----------------	---	---	---	---	---	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

działanie:

- PC ← PC + 2
- (SP) ← (SP) + 1
- ((SP)) ← (PC₇₋₀)
- (SP) ← (SP) + 1
- ((SP)) ← (PC₁₅₋₈)
- (PC₁₀₋₀) ← adr_11

zmiana flag: -

przykład: Po uruchomieniu mikrokontrolera, SP jest ustawiany na wartość 07h. Jeżeli etykieta *podprogram* wskazuje na podprogram umieszczony pod adresem 0345h, to po wykonaniu instrukcji:

ACALL podprogram

umieszczonej pod adresem 0123h, licznik rozkazów, PC, przyjmie wartość 0345h a wskaźnik stosu, SP, wartość 09h. W pamięci RAM, pod adresami 08h i 09h będą umieszczone bajty o wartościach, odpowiednio, 25h i 01h.

uwagi: Instrukcja działa w obrębie jednej strony pamięci programu (patrz rys.1.2.10.) i jest wykonywana na stronie, w której znajduje się następna instrukcja po ACALL. Jeżeli podprogram jest ulokowany na innej stronie, wywołanie podprogramu spowoduje uruchomienie kodu opisanego 11-bitowym adre-

sem bieżącej strony. Dobry kompilator powinien rozpoznać ten problem i wykreować komunikat błędu.

ADD A, <src>	wykonywanie dodawania (ang. add)
<p>opis: Do zawartości akumulatora dodawany jest wskazany bajt a wynik operacji jest umieszczany w akumulatorze. Operacja dodawania wpływa na stan znaczników CY, AC i OV.</p> <p>Flagi CY i AC są ustawiane gdy nastąpiło przepełnienie wyniku na pozycjach z indeksami, odpowiednio, 7 i 4. Przy braku przepełnienia wskaźniki są kasowane. Jeżeli dodawane są liczby całkowite bez znaku to ustawienie flagi CY oznacza przepełnienie.</p> <p>Flaga OV jest ustawiana jeżeli przepełnienie występuje na pozycji 6 (ale nie na pozycji 7) lub na pozycji 7 (ale nie na pozycji 6). W każdym innym przypadku flaga OV jest zerowana. Przy dodawaniu liczb ze znakiem, flaga OV wskazuje pojawienie się liczby ujemnej przy dodawaniu dwu liczb dodatnich lub pojawienie się liczby dodatniej przy dodawaniu dwu liczb ujemnych.</p> <p>Instrukcja ADD obejmuje 4 sposoby adresowania argumentu.</p> <p>przykład: Jeżeli akumulator zawiera daną o wartości 0C3h (11000011b) a w rejestrze R0 jest dana o wartości 0AAh (10101010b) to instrukcja:</p> <p style="text-align: center;">ADD A,R0</p> <p>spowoduje, że w akumulatorze pojawi się dana o wartości 6Dh (01101101b), flaga AC będzie wyzerowana a flagi CY oraz OV będą ustawione w stan jedynki logicznej.</p>	

ADD A, Rn	dodawanie								
<p>opis: Do zawartości akumulatora dodawany jest stan rejestru roboczego R0..R7.</p> <p>bajty/cykle: 1/1</p> <p>kodowanie: <table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 5px;">0</td> <td style="padding: 2px 5px;">0</td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;">0</td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;">r_0</td> <td style="padding: 2px 5px;">r_1</td> <td style="padding: 2px 5px;">r_2</td> </tr> </table></p> <p>działanie: $(A) \leftarrow (A) + (Rn)$ gdzie $Rn \in \{R0, \dots, R7\}$</p> <p>zmiana flag: CY, AC i OV</p>		0	0	1	0	1	r_0	r_1	r_2
0	0	1	0	1	r_0	r_1	r_2		

przykład: ADD A, R5

ADD A, adr	dodawanie
-------------------	-----------

opis: Do zawartości akumulatora dodawany jest stan rejestru umieszczonego pod adresem bezpośrednim, oznaczonym etykietą: *adr*.

bajty/cykle: 2/1

kodowanie:

0 0 1 0	0 1 0 1
---------	---------

adr

działanie: $(A) \leftarrow (A) + (\text{adr})$

zmiana flag: CY, AC i OV

przykład: ADD A, adres
ADD A, 123

ADD A, @Ri	dodawanie
-------------------	-----------

opis: Do zawartości akumulatora dodawany jest stan rejestru umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1.

bajty/cykle: 1/1

kodowanie:

0 0 1 0	0 1 1 i
---------	---------

 gdzie $i \in \{0,1\}$

działanie: $(A) \leftarrow (A) + ((Ri))$ gdzie $Ri \in \{R0, R1\}$

zmiana flag: CY, AC i OV

przykład: ADD A, @R1

ADD A, #n	dodawanie
------------------	-----------

opis: Do zawartości akumulatora dodawany jest bajt o wartości *n*.

bajty/cykle: 2/1

kodowanie:

0 0 1 0	0 1 0 0
---------	---------

n

działanie: $(A) \leftarrow (A) + n$

zmiana flag: CY, AC i OV

przykład: ADD A, #123

ADDC A, <src>	wykonywanie dodawania z przeniesieniem (ang. add with carry)
<p>opis: Do zawartości akumulatora dodawany jest wskazany bajt oraz bit CY a wynik operacji jest umieszczany w akumulatorze. Operacja dodawania wpływa na stan znaczników CY, AC i OV.</p> <p>Flagi CY i AC są ustawiane gdy nastąpiło przepełnienie wyniku na pozycjach z indeksami, odpowiednio, 7 i 4. Przy braku przepełnienia wskaźniki są kasowane. Jeżeli dodawane są liczby całkowite bez znaku to ustawienie flagi CY oznacza przepełnienie.</p> <p>Flaga OV jest ustawiana jeżeli przepełnienie występuje na pozycji 6 (ale nie na pozycji 7) lub na pozycji 7 (ale nie na pozycji 6). W każdym innym przypadku flaga OV jest zerowana. Przy dodawaniu liczb ze znakiem, flaga OV wskazuje pojawienie się liczby ujemnej przy dodawaniu dwu liczb dodatnich lub pojawienie się liczby dodatniej przy dodawaniu dwu liczb ujemnych.</p> <p>Instrukcja ADDC obejmuje 4 sposoby adresowania argumentu.</p> <p>przykład: Jeżeli akumulator zawiera daną o wartości 0C3h (11000011b); w rejestrze R0 jest dana o wartości 0AAh (10101010b) a wartość CY=1, to instrukcja:</p> <p style="text-align: center;">ADDC A,R0</p> <p>spowoduje, że w akumulatorze pojawi się dana o wartości 6Eh (01101110b), flaga AC będzie wyzerowana a flagi CY oraz OV będą ustawione w stan jedynki logicznej.</p>	

ADDC A, Rn	dodawanie z przeniesieniem
-------------------	----------------------------

- opis:** Do zawartości akumulatora dodawany jest stan rejestru roboczego R0..R7 oraz stan bitu CY.
- bajty/cykle:** 1/1
- kodowanie:**

0 0 1 1	1 r ₀ r ₁ r ₂
---------	--
- działanie:** $(A) \leftarrow (A) + (Rn) + (CY)$ gdzie $Rn \in \{R0, \dots, R7\}$
- zmiana flag:** CY, AC i OV
- przykład:** ADDC A, R5

ADDC A, adr

dodawanie z przeniesieniem

opis: Do zawartości akumulatora dodawany jest stan rejestru umieszczonego pod adresem bezpośrednim, oznaczonym etykietą *adr*, oraz stan bitu CY.

bajty/cykle: 2/1

kodowanie:

0 0 1 1	0 1 0 1		adr
---------	---------	--	-----

działanie: $(A) \leftarrow (A) + (\text{adr}) + (\text{CY})$

zmiana flag: CY, AC i OV

przykład: ADDC A, adres
ADDC A, 123

ADDC A, @Ri

dodawanie z przeniesieniem

opis: Do zawartości akumulatora dodawany jest stan rejestru umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1 oraz stan bitu CY.

bajty/cykle: 1/1

kodowanie:

0 0 1 1	0 1 1 i		gdzie $i \in \{0, 1\}$
---------	---------	--	------------------------

działanie: $(A) \leftarrow (A) + ((Ri)) + (\text{CY})$ gdzie $Ri \in \{R0, R1\}$

zmiana flag: CY, AC i OV

przykład: ADDC A, @R1

ADDC A, #n

dodawanie z przeniesieniem

opis: Do zawartości akumulatora dodawany jest bajt o wartości *n* oraz stan bitu CY.

bajty/cykle: 2/1

kodowanie:

0 0 1 1	0 1 0 0		n
---------	---------	--	---

działanie: $(A) \leftarrow (A) + n + (\text{CY})$

zmiana flag: CY, AC i OV

przykład: ADDC A, #123

AJMP adr_11skok bezwarunkowy
(ang. absolute jump)

opis: Przekazuje wykonywanie programu pod adres określony słowem 11-bitowym.

bajty/cykle: 2/2

kodowanie:

a ₁₀ a ₉ a ₈ 0	0 0 0 1	a ₇ a ₆ a ₅ a ₄	a ₃ a ₂ a ₁ a ₀
---	---------	---	---

działanie: PC ← PC + 2

PC₁₀₋₀ ← adres11

zmiana flag: -

przykład: Jeżeli instrukcja AJMP znajduje się pod adresem 0345h a etykieta *inny_program* wskazuje na kod umieszczony pod adresem 0125h, to po wykonaniu instrukcji:

```
AJMP inny_program
```

licznik rozkazów, PC, przyjmie wartość 0125h.

uwagi: Instrukcja działa w obrębie jednej strony pamięci programu (patrz rys.1.2.10.) i jest wykonywana na stronie, w której znajduje się następna instrukcja po AJMP. Jeżeli adres docelowy skoku jest ulokowany na innej stronie, wykonanie instrukcji spowoduje uruchomienie kodu opisanego 11-bitowym adresem bieżącej strony. Dobry kompilator powinien rozpoznać ten problem i wykreować komunikat błędu.

ANL <dest >, <src >iloczyn logiczny bajtów
(ang. logical AND for byte variables)

opis: Instrukcja wykonuje operację iloczynu logicznego AND na odpowiadających sobie bitach dwu bajtów, umieszczonych pod adresami <dest> i <src>. Wynik operacji jest umieszczony pod adresem <dest>. Instrukcja iloczynu logicznego bajtów obejmuje 6 sposobów adresowania argumentu.

przykład: Jeżeli akumulator zawiera daną o wartości 0C3h (11000011b) a w rejestrze R0 jest dana o wartości 55h (01010101b) to wykonanie instrukcji:

```
ANL A,R0
```

wprowadza do akumulatora wartość 41h (01000001b).

uwaga: Jeżeli instrukcja jest użyta do modyfikacji stanu portów P0..P3, daną do wykonania instrukcji jest stan rejestrów linii

portu a nie stan końcówek portu. Wynik operacji jest zapisywany do rejestru portu i może wpływać na stan końcówek portu.

ANL A, Rn

iloczyn logiczny bajtów

opis: Instrukcja wykonuje operację AND pomiędzy bitami akumulatora a bitami rejestru roboczego R0..R7.

bajty/cykle: 1/1

kodowanie:

0	1	0	1
---	---	---	---

 |

1	r_0	r_1	r_2
---	-------	-------	-------

działanie: $(A) \leftarrow (A) \wedge (Rn)$ gdzie $Rn \subset \{R0, \dots, R7\}$

zmiana flag: -

przykład: ANL A, R5

ANL A, adr

iloczyn logiczny bajtów

opis: Instrukcja wykonuje operację AND pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą *adr*.

bajty/cykle: 2/1

kodowanie:

0	1	0	1
---	---	---	---

 |

0	1	0	1
---	---	---	---

 |

adr

działanie: $(A) \leftarrow (A) \wedge (adr)$

zmiana flag: -

przykład: ANL A, adres
ANL A, 123

ANL A, @Ri

iloczyn logiczny bajtów

opis: Instrukcja wykonuje operację AND pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1.

bajty/cykle: 1/1

kodowanie:

0	1	0	1
---	---	---	---

 |

0	1	1	i
---	---	---	-----

 gdzie $i \in \{0, 1\}$

działanie: $(A) \leftarrow (A) \wedge ((Ri))$ gdzie $Ri \subset \{R0, R1\}$

zmiana flag: -

przykład: ANL A,@R1

ANL A, #n

iloczyn logiczny bajtów

opis: Instrukcja wykonuje operację AND pomiędzy bitami akumulatora a bitami bajtu o wartości n .

bajty/cykle: 2/1

kodowanie:

0 1 0 1	0 1 0 0	n
---------	---------	---

działanie: $(A) \leftarrow (A) \wedge n$

zmiana flag: -

przykład: ANL A, #123

ANL adr, A

iloczyn logiczny bajtów

opis: Instrukcja wykonuje operację AND pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą adr . Wynik operacji jest umieszczony pod adresem adr .

bajty/cykle: 2/1

kodowanie:

0 1 0 1	0 0 1 0	adr
---------	---------	-----

działanie: $(adr) \leftarrow (adr) \wedge (A)$

zmiana flag: -

przykład: ANL adr, A
ANL 123, A

ANL adr, #n

iloczyn logiczny bajtów

opis: Instrukcja wykonuje operację AND pomiędzy bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą adr a bitami bajtu o wartości n . Wynik operacji jest umieszczany pod adresem adr .

bajty/cykle: 3/2

kodowanie:

0 1 0 1	0 0 1 1	adr	n
---------	---------	-----	---

działanie: $(adr) \leftarrow (adr) \wedge n$

zmiana flag: -

przykład: ANL adr, #123
ANL 123, #123

ANL C, <src_bit >	iloczyn logiczny bitów (ang. logical AND for bit variables)
<p>opis: Instrukcja wykonuje operację iloczynu logicznego AND pomiędzy bitem CY i bitem źródła, wskazywanego adresem bezpośrednim. Znak "/", poprzedzający adres bitu źródła oznacza, że instrukcja wykona operację iloczynu na negacji bitu źródła. Instrukcja iloczynu logicznego bitów obejmuje 2 sposoby adresowania bitu argumentu.</p> <p>przykład: W celu określenia, czy spełniony jest warunek: P1.0=1, ACC.7=1 oraz OV=0 trzeba wykonać 3 instrukcje kodu:</p> <pre>MOV C, P1.0 ANL C, ACC.7 ANL C, /OV</pre>	

ANL C, bit	iloczyn logiczny bitów			
<p>opis: Instrukcja wykonuje operację AND pomiędzy bitem CY i bitem źródła.</p> <p>bajty/cykle: 2/2</p> <p>kodowanie: <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="padding: 2px 10px;">1 0 0 0</td><td style="padding: 2px 10px;">0 0 1 0</td></tr></table> <table border="1" style="display: inline-table; vertical-align: middle; margin-left: 20px;"><tr><td style="padding: 2px 10px;">adres bitu</td></tr></table></p> <p>działanie: $(CY) \leftarrow (CY) \wedge (\text{bit})$</p> <p>zmiana flag: CY</p> <p>przykład: ANL C, TF0 ANL C, 08Dh</p>	1 0 0 0	0 0 1 0	adres bitu	
1 0 0 0	0 0 1 0			
adres bitu				

ANL C, /bit	iloczyn logiczny bitów			
<p>opis: Instrukcja wykonuje operację AND pomiędzy bitem CY i negacją bitu źródła.</p> <p>bajty/cykle: 2/2</p> <p>kodowanie: <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="padding: 2px 10px;">1 0 1 1</td><td style="padding: 2px 10px;">0 0 0 0</td></tr></table> <table border="1" style="display: inline-table; vertical-align: middle; margin-left: 20px;"><tr><td style="padding: 2px 10px;">adres bitu</td></tr></table></p>	1 0 1 1	0 0 0 0	adres bitu	
1 0 1 1	0 0 0 0			
adres bitu				

działanie: $(CY) \leftarrow (CY) \wedge \text{/}(bit)$
zmiana flag: CY
przykład: ANL C, /TF0
 ANL C, /08Dh

CJNE <dest>, <src>, d	porównywanie argumentów i skok względny gdy nierówne (ang. compare and jump if not equal)
--	---

opis: Instrukcja porównuje wartości argumentów, które są wskazywane adresami <dest> i <src>. W przypadku braku równości, do licznika PC jest dodawana wartość przesunięcia względnego *d* - jest wykonywany skok względny. Gdy argumenty są sobie równe, wykonywana jest kolejna instrukcja kodu. Przesunięcie jest liczbą ze znakiem zapisaną w kodzie U2 i jest określone względem adresu następnej instrukcji po CJNE.

W ramach wykonywania instrukcji, zmianom ulega stan bitu CY - jeżeli wartość pierwszego z argumentów, <dest>, jest mniejsza od wartości następnego, <src> to bit CY jest ustawiany. W przeciwnym przypadku bit CY jest zerowany.

Instrukcja CJNE obejmuje 4 sposoby adresowania argumentów.

przykład: Jeżeli rejestr R7 zawiera daną o wartości 56h i będzie wykonana sekwencja kodu:

```

      CJNE R7, #60h, dalej_1    ; linia 1
      ....
dalej_1: JC    dalej_2          ; linia 2
      ....
dalej_2: ....                  ; linia 3
      ANL  C, ACC.7
```

to po wykonaniu instrukcji w linii 1 nastąpi skok do linii 2 ponieważ $(R7) \neq 60$; w linii 2 nastąpi skok do linii 3 ponieważ $(R7) < 60$.

CJNE A, adr, d	porównywanie i skok gdy brak równości
-----------------------	---------------------------------------

opis: Instrukcja porównuje stan akumulatora ze stanem rejestru umieszczonego pod adresem bezpośrednim *adr*.

bajty/cykle: 3/2**kodowanie:**

1 0 1 1 0 1 0 1

adr

d

działanie: $(PC) \leftarrow (PC) + 3$
gdy $(A) \neq (\text{adr})$ to $(PC) = (PC) + d$
gdy $(A) < (\text{adr})$ to $CY = 1$
gdy $(A) \geq (\text{adr})$ to $CY = 0$ **zmiana flag:** CY**przykład:** CJNE A, TMOD, dalej
CJNE A, 089h, dalej**CJNE A, #n, d**porównywanie i skok gdy
brak równości**opis:** Instrukcja porównuje stan akumulatora z bajtem o wartości n .**bajty/cykle:** 3/2**kodowanie:**

1 0 1 1 0 1 0 0

n

d

działanie: $(PC) \leftarrow (PC) + 3$
gdy $(A) \neq n$ to $(PC) = (PC) + d$
gdy $(A) < n$ to $CY = 1$
gdy $(A) \geq n$ to $CY = 0$ **zmiana flag:** CY**przykład:** CJNE A, #dana, dalej
CJNE A, #123, dalej**CJNE Rn, #n, d**porównywanie i skok gdy
brak równości**opis:** Instrukcja porównuje stan rejestru R0..R7 z bajtem o wartości n .**bajty/cykle:** 3/2**kodowanie:**

1 0 1 1 1 r ₀ r ₁ r ₂
--

n

d

działanie: $(PC) \leftarrow (PC) + 3$
gdy $(Rn) \neq n$ to $(PC) = (PC) + d$
gdy $(Rn) < n$ to $CY = 1$
gdy $(Rn) \geq n$ to $CY = 0$ **zmiana flag:** CY**przykład:** CJNE R0, #dana, dalej

CJNE R5, #123, dalej

CJNE @Ri, #n, dporównywanie i skok gdy
brak równości**opis:** Instrukcja porównuje stan bajtu adresowanego pośrednio przez rejestr R0 lub R1 z bajtem o wartości n .**bajty/cykle:** 3/2**kodowanie:**

1	0	1	1	0	1	1	i
---	---	---	---	---	---	---	---

n

d

działanie: $(PC) \leftarrow (PC) + 3$
gdy $((Ri)) \neq n$ to $(PC) = (PC) + d$ gdzie $i \in \{0, 1\}$
gdy $((Ri)) < n$ to $CY = 1$
gdy $((Ri)) \geq n$ to $CY = 0$ **zmiana flag:** CY**przykład:** CJNE @R0, #dana, dalej
CJNE @R1, #123, dalej**CLR A**kasowanie akumulatora
(ang. clear accumulator)**opis:** Instrukcja kasuje wszystkie bity akumulatora - kasuje stan akumulatora.**bajty/cykle:** 1/1**kodowanie:**

1	1	1	0
---	---	---	---

0	1	0	0
---	---	---	---

działanie: $(A) \leftarrow 0$ **zmiana flag:** -**CLR bit**

kasowanie bitu

opis: Instrukcja wykonuje operację kasowania bitu wskazywanego adresem bezpośrednim.**bajty/cykle:** 1/1**kodowanie:**

1	1	0	0
---	---	---	---

0	0	1	0
---	---	---	---

działanie: $(bit) \leftarrow 0$ **zmiana flag:** -

przykład: Jeżeli stan rejestru portu P1 był równy 5Dh (01011101b), to po wykonaniu operacji:

CLR P1.2

będzie on wynosił 59h (0101001b).

CLR C

kasowanie bitu CY

opis: Instrukcja wykonuje operację kasowania bitu CY.

bajty/cykle: 1/1

kodowanie:

1 1 0 0	0 0 1 1
---------	---------

działanie: (CY) ← 0

zmiana flag: CY

CPL A

negowanie stanu akumulatora
(ang. complement accumulator)

opis: Instrukcja powoduje operację zanegowania stanu wszystkich bitów akumulatora (uzupełnienie do 1).

bajty/cykle: 1/1

kodowanie:

1 1 1 1	0 1 0 0
---------	---------

działanie: (A) ← not (A)

zmiana flag: -

CPL bit

negowanie bitu
(ang. complement bit)

opis: Instrukcja wykonuje operację zanegowania stanu bitu wskazywanego adresem bezpośrednim.

bajty/cykle: 1/1

kodowanie:

1 0 1 1	0 0 1 0
---------	---------

działanie: (bit) ← not (bit)

zmiana flag: -

przykład: Jeżeli stan rejestru portu P1 był równy 5Dh (01011101b), to po wykonaniu operacji:

CPL P1.2

będzie on wynosił 59h (0101001b).

uwaga: Jeżeli instrukcja jest użyta do modyfikacji stanu portów P0..P3, daną do wykonania instrukcji jest stan rejestrów linii portu a nie stan końcówek portu. Wynik operacji jest zapisywany do rejestru portu i może wpływać na stan końcówek portu.

CPL C

negowanie bitu CY

opis: Instrukcja wykonuje operację zanegowania stanu bitu CY.

bajty/cykle: 1/1

kodowanie:

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

zmiana flag: CY

DA Akorekcja dziesiętna
(ang. decimal adjust)

opis: Instrukcja wykonuje korektę stanu akumulatora po operacji dodawania dwu bajtów zawierających liczby zapisane w kodzie BCD (liczby dziesiętne). W każdym z dodawanych bajtów znajdują się dwie liczby dziesiętne, młodsza i starsza. Młodszej liczbie przypisane są bity z indeksami 0..3 a starszej 4..7.

Dodawanie dwu bajtów z liczbami dziesiętnymi jest realizowane standardową operacją dodawania binarnego, ADD lub ADDC. Operacja dodawania zmienia stan akumulatora i flag, CY i AC. Po wykonaniu operacji dodawania binarnego, uzyskany wynik trzeba skorygować dziesiętnie w celu uzyskania poprawnej formy zapisu wyniku w formacie BCD. W tym celu, po każdej instrukcji dodawania, należy dodatkowo wykonać korektę dziesiętną uzyskanego wyniku:

```
ADD A, #79h
DA A
```

Po wykonaniu korekcji dziesiętnej wyniku dodawania, w przypadku gdy liczba dziesiętna jest większa od 99, ustawiana jest flaga CY. Operacja nie zmienia wskaźników AC i OV.

Instrukcja korekcji dziesiętnej działa poprawnie wyłącznie w powiązaniu z operacją dodawania. Nie nadaje się ona do prostej konwersji stanu akumulatora na kod BCD - konwersję można przeprowadzić jedynie dla liczby z zakresu 0..15. In-

strukcja nie może być stosowana do korekcji wyniku odejmowania dwu liczb dziesiętnych.

bajty/cykle: 1/1

kodowanie:

1	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

działanie: gdy $((A_{3..0}) > 9) \vee ((AC) = 1)$ to $(A_{3..0}) = (A_{3..0}) + 6$
oraz
gdy $((A_{7..4}) > 9) \vee ((CY) = 1)$ to $(A_{7..4}) = (A_{7..4}) + 6$

zmiana flag: CY

DEC <src>	zmniejszanie stanu o 1 (ang. decrement)
<p>opis: Stan wskazanego bajtu jest zmniejszany o wartość 1. Stan bajtu jest traktowany jako liczba zapisana w naturalnym kodzie binarnym. Wykonanie instrukcji DEC na bajcie o wartości 00h powoduje otrzymanie liczby 0FFh. Instrukcja zmniejszania stanu, DEC, obejmuje 4 sposoby adresowania argumentu.</p> <p>uwaga: Jeżeli instrukcja jest użyta do modyfikacji stanu portów P0..P3, daną do wykonania instrukcji jest stan rejestrów linii portu a nie stan końcówek portu. Wynik operacji jest zapisywany do rejestru portu i może wpływać na stan końcówek portu.</p>	

DEC A	zmniejszanie o 1
--------------	------------------

opis: Zmniejszaniu podlega stan akumulatora.

bajty/cykle: 1/1

kodowanie:

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---

działanie: $(A) \leftarrow (A) - 1$

zmiana flag: -

DEC Rn	zmniejszanie o 1
---------------	------------------

opis: Zmniejszaniu podlega stan wybranego rejestru R0..R7.

bajty/cykle: 1/1

kodowanie:

0 0 0 1	1 r ₀ r ₁ r ₂
---------	--

działanie: $(R_n) \leftarrow (R_n) - 1$ gdzie $R_n \in \{R_0, \dots, R_7\}$

zmiana flag: -

przykład: DEC R5

DEC adr

zmniejszanie o 1

opis: Zmniejszaniu podlega stan bajtu adresowanego bezpośrednio.

bajty/cykle: 2/1

kodowanie:

0 0 0 1	0 1 0 1
---------	---------

adr

działanie: $(adr) \leftarrow (adr) - 1$

zmiana flag: -

przykład: DEC P1

DEC @Ri

zmniejszanie o 1

opis: Zmniejszaniu podlega stan bajtu umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1.

bajty/cykle: 1/1

kodowanie:

0 0 0 1	0 1 1 i
---------	---------

 gdzie $i \in \{0,1\}$

działanie: $((R_i)) \leftarrow ((R_i)) - 1$ gdzie $R_i \in \{R_0, R_1\}$

zmiana flag: -

przykład: DEC @R1

DIV AB

dzielenie
(ang. divide)

opis: Instrukcja wykonuje operację dzielenia dwu liczb 8-bitowych bez znaku - zawartość akumulatora jest dzielona przez stan rejestru B. Część całkowita wyniku dzielenia jest wpisywana do akumulatora a reszta do rejestru B. Instrukcja kasuje flagi CY oraz OV.

uwaga: Jeżeli stan rejestru B jest zerem (00h) to po wykonaniu instrukcji stan akumulatora i rejestru B są nieokreślone - flaga

CY jest kasowana a flaga OV ustawiana w stan jedynek logicznej.

bajty/cykle: 1/4

kodowanie:

1 0 0 0	0 1 0 0
---------	---------

działanie: (A) ← część całkowita z [(A)/(B)]

(B) ← reszta z [(A)/(B)]

zmiana flag: CY=0, OV=0 lub CY=0, OV=1 przy dzieleniu przez 0

przykład: Jeżeli stan akumulatora jest równy 251 (0FBh lub 11111011b) a stan rejestru B wynosi 18 (12h lub 00010010b) to po wykonaniu operacji dzielenia:

DIV A/B

w akumulatorze znajdzie się liczba 13 (0Dh lub 00001101b)

a w rejestrze B liczba 17 (11h lub 00010111b) ponieważ

$$13 \cdot 18 + 17 = 251$$

DJNZ <dest>, d	zmniejszanie o 1, porównywanie argumentów i skok względny gdy nie zero (ang. decrement and jump if not zero)
-----------------------------	--

opis: Instrukcja zmniejsza stan wskazanego argumentu o 1 i próbuje wynik operacji z zerem. Jeżeli argument nie jest zerem, do licznika rozkazów PC jest dodawane przesunięcie *d* - wykonywany jest skok względny. Przesunięcie *d* jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po DJNZ. Argument może być rejestrem roboczym, R0..R7, lub rejestrem adresowanym bezpośrednio.

Instrukcja DJNZ obejmuje 2 sposoby adresowania argumentu. Wykonanie instrukcji nie zmienia stanu flag.

uwaga: Jeżeli instrukcja jest użyta do modyfikacji stanu portów P0..P3, daną do wykonania instrukcji jest stan rejestrów linii portu a nie stan końcówek portu. Wynik operacji jest zapisywany do rejestru portu i może wpływać na stan końcówek portu.

przykład: Instrukcja pozwala na wykonanie prostej pętli o zadanej liczbie powtórzeń. Zestaw instrukcji:

```
MOV   R2, #8
xxx:  CPL   P1.7
      DJNZ  A,R0
```

pozwała na ośmiokrotną zmianę stanu linii P1.7 portu P1.

DJNZ Rn,dzmniejszanie, porównywanie
i skok względny gdy nie zero**opis:** Zmniejszaniu podlega stan wybranego rejestru R0..R7.**bajty/cykle:** 2/2**kodowanie:**

1 1 0 1	1 r ₀ r ₁ r ₂	d
---------	--	---

działanie: $(PC) \leftarrow (PC) + 2$
 $(Rn) \leftarrow (Rn) - 1$
gdy $(Rn) \neq 0$ to $(PC) = (PC) + d$ **zmiana flag:** -**przykład:** DJNZ R5, PETA**DJNZ adr,d**zmniejszanie, porównywanie
i skok względny gdy nie zero**opis:** Zmniejszaniu podlega stan bajtu pamięci, który jest adresowany bezpośrednio.**bajty/cykle:** 3/2**kodowanie:**

1 1 0 1 0 1 0 1	adr	d
-----------------	-----	---

działanie: $(PC) \leftarrow (PC) + 2$
 $(adr) \leftarrow (adr) - 1$
gdy $(adr) \neq 0$ to $(PC) = (PC) + d$ **zmiana flag:** -**przykład:** DJNZ 123, petla**INC <src>**zwiększanie stanu o 1
(ang. increment)**opis:** Stan wskazanego bajtu jest zwiększany o wartość 1. Stan bajtu jest traktowany jako liczba zapisana w naturalnym kodzie binarnym. Wykonanie instrukcji INC na bajcie o wartości 0FFh powoduje otrzymanie liczby 00h. Instrukcja zwiększania stanu, INC, obejmuje 5 sposobów adresowania argumentu.**uwaga:** Jeżeli instrukcja jest użyta do modyfikacji stanu portów P0..P3, daną do wykonania instrukcji jest stan rejestrów linii portu a nie stan końcówek portu.

Wynik operacji jest zapisywany do rejestru portu i może wpływać na stan końcówek portu.

INC A

zwiększanie o 1

opis: Zwiększaniu podlega stan akumulatora.

bajty/cykle: 1/1

kodowanie:

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

działanie: $(A) \leftarrow (A) + 1$

zmiana flag: -

INC Rn

zwiększanie o 1

opis: Zwiększaniu podlega stan wybranego rejestru R0..R7.

bajty/cykle: 1/1

kodowanie:

0	0	0	0	1	r_0	r_1	r_2
---	---	---	---	---	-------	-------	-------

działanie: $(Rn) \leftarrow (Rn) + 1$ gdzie $Rn \in \{R0, \dots, R7\}$

zmiana flag: -

przykład: INC R5

INC adr

zwiększanie o 1

opis: Zwiększaniu podlega stan bajtu adresowanego bezpośrednio.

bajty/cykle: 2/1

kodowanie:

0	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---

adr

działanie: $(adr) \leftarrow (adr) + 1$

zmiana flag: -

przykład: INC P1

INC @Ri

zwiększanie o 1

opis: Zwiększaniu podlega stan bajtu umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1.

bajty/cykle: 1/1

kodowanie:

0 0 0 0	0 1 1 i
---------	---------

gdzie $i \in \{0,1\}$

działanie: $((Ri)) \leftarrow ((Ri)) + 1$

gdzie $Ri \in \{R0, R1\}$

zmiana flag: -

przykład: INC @R1

INC DPTR

zwiększanie o 1 stanu wskaźnika danych
(ang. increment data pointer)

opis: Zwiększaniu o wartość 1 podlega stan 16-bitowego rejestru DPTR

bajty/cykle: 1/2

kodowanie:

1 0 1 0	0 0 1 1
---------	---------

działanie: $(DPTR) \leftarrow (DPTR) + 1$

zmiana flag: -

JB bit,d

skok gdy bit ustawiony
(ang. jump if bit set)

opis: Instrukcja wykonuje operację skoku w przypadku, gdy bit wskazywany w sposób bezpośredni ma wartość jedynki logicznej. W takim przypadku, do licznika rozkazów PC jest dodawane przesunięcie d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po JB. Wykonanie instrukcji nie zmienia stanu flag.

bajty/cykle: 3/2

kodowanie:

0 0 1 0	0 0 0 0	bit	d
---------	---------	-----	---

działanie: $(PC) \leftarrow (PC) + 3$

gdy $(bit) = 1$ to $(PC) = (PC) + d$

zmiana flag: -

przykład: JB P1.2, dalej

uwaga: Wskazywany instrukcją bit nie jest zmieniany - jest on tylko testowany.

JBC bit,d

skok i kasowanie bitu gdy
bit ustawiony
(ang. jump if bit is set and
clear bit)

opis: Instrukcja wykonuje operację skoku w przypadku, gdy bit wskazywany w sposób bezpośredni ma wartość jedynki logicznej. W takim przypadku, bit jest zerowany a do licznika rozkazów PC jest dodawane przesunięcie d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po JBC. Wykonanie instrukcji nie zmienia stanu flag.

bajty/cykle: 3/2

kodowanie:

0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

bit

d

działanie: $(PC) \leftarrow (PC) + 3$
gdy (bit) = 1 to
(bit) \leftarrow 0
(PC) \leftarrow (PC) + d

zmiana flag: -

przykład: JB P1.2, dalej

uwaga: Jeżeli instrukcja jest użyta do testowania stanu wybranej linii portów P0..P3, daną do wykonania instrukcji jest wskazany instrukcją stan rejestru linii portu a nie stan końcówki tej linii. Po wykonaniu operacji, rejestr wskazanej linii portu jest zerowany.

JC d

skok gdy bit CY ustawiony
(ang. jump if carry is set)

opis: Instrukcja wykonuje operację skoku w przypadku, gdy bit CY ma wartość jedynki logicznej. W takim przypadku, do licznika rozkazów PC jest dodawane przesunięcie d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po JC. Wykonanie instrukcji nie zmienia stanu flag.

bajty/cykle: 2/2

kodowanie:

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

d

działanie: $(PC) \leftarrow (PC) + 2$
 gdy $(CY) = 1$ to $(PC) = (PC) + d$

zmiana flag: -

przykład: JC dalej

JMP @A+DPTR

skok pośredni
 (ang. jump indirect)

opis: Instrukcja dodaje do 16-bitowego rejestru DPTR stan akumulatora a wynik dodawania umieszcza w rejestrze PC - jest wykonywany skok pod adres PC. Stan akumulatora jest traktowany jako naturalna liczba binarna. Instrukcja nie zmienia stanu akumulatora, stanu wskaźnika DPTR oraz stanu flag.

bajty/cykle: 1/2

kodowanie:

0 1 1 1	0 0 1 1
---------	---------

działanie: $(PC) \leftarrow (A) + (DPTR)$

zmiana flag: -

przykład: Jeżeli stan akumulatora jest liczbą parzystą z zakresu 0..6, to po wykonaniu podanego zestawu instrukcji, dalsze wykonanie programu zostanie przekazane pod jeden z adresów, prog_0 .. prog_3.

```

MOV    DPTR, #xxx
JMP    @A+DPTR
xxx:
AJAMP  prog_0
AJAMP  prog_1
AJAMP  prog_2
AJAMP  prog_3

```

JNB bit,d

skok gdy bit skasowany
 (ang. jump if bit not set)

opis: Instrukcja wykonuje operację skoku w przypadku, gdy bit wskazywany w sposób bezpośredni ma wartość zera logicznego. W takim przypadku, do licznika rozkazów PC jest dodawane przesunięcie d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po JNB. Wykonanie instrukcji nie zmienia stanu flag.

bajty/cykle: 3/2

kodowanie:	<table border="1"><tr><td>0 0 1 1 0 0 0 0</td></tr></table>	0 0 1 1 0 0 0 0	<table border="1"><tr><td>bit</td></tr></table>	bit	<table border="1"><tr><td>d</td></tr></table>	d
0 0 1 1 0 0 0 0						
bit						
d						
działanie:	(PC) ← (PC) + 3 gdy (bit) = 0 to (PC) = (PC) + d					
zmiana flag:	-					
przykład:	JNB 21, dalej_1 JNB ACC.0, dalej_2					
uwaga:	Wskazywany instrukcją bit nie jest zmieniany - jest on tylko testowany.					

JNC d	skok gdy bit CY skasowany (ang. jump if carry not set)
--------------	---

opis: Instrukcja wykonuje operację skoku w przypadku, gdy bit CY ma wartość zera logicznego. W takim przypadku, do licznika rozkazów PC jest dodawane przesunięcie d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po JNC. Wykonanie instrukcji nie zmienia stanu flag.

bajty/cykle: 2/2

kodowanie:	<table border="1"><tr><td>0 1 0 1</td><td>0 0 0 0</td></tr></table>	0 1 0 1	0 0 0 0	<table border="1"><tr><td>d</td></tr></table>	d
0 1 0 1	0 0 0 0				
d					
działanie:	(PC) ← (PC) + 2 gdy (CY) = 0 to (PC) = (PC) + d				
zmiana flag:	-				
przykład:	JNC dalej				

JNZ d	skok gdy stan akumulatora różny od zera (ang. jump if accumulator not zero)
--------------	--

opis: Instrukcja wykonuje operację skoku w przypadku, gdy stan akumulatora jest różny od zera. W takim przypadku, do licznika rozkazów PC jest dodawane przesunięcie d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po JNZ. Instrukcja nie zmienia stanu akumulatora ani stanu flag.

bajty/cykle: 2/2

kodowanie:	<table border="1"><tr><td>0 1 1 1</td><td>0 0 0 0</td></tr></table>	0 1 1 1	0 0 0 0	<table border="1"><tr><td>d</td></tr></table>	d
0 1 1 1	0 0 0 0				
d					
działanie:	(PC) ← (PC) + 2 gdy (A) = 1 to (PC) = (PC) + d				
zmiana flag:	-				
przykład:	JNZ dalej				

JZ d	skok gdy stan akumulatora równy zero (ang. jump if accumulator zero)
-------------	---

opis: Instrukcja wykonuje operację skoku w przypadku, gdy stan akumulatora jest równy zero. W takim przypadku, do licznika rozkazów PC jest dodawane przesunięcie d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po JZ. Instrukcja nie zmienia stanu akumulatora ani stanu flag.

bajty/cykle: 2/2

kodowanie:	<table border="1"><tr><td>0 1 1 0</td><td>0 0 0 0</td></tr></table>	0 1 1 0	0 0 0 0	<table border="1"><tr><td>d</td></tr></table>	d
0 1 1 0	0 0 0 0				
d					
działanie:	(PC) ← (PC) + 2 gdy (A) = 0 to (PC) = (PC) + d				
zmiana flag:	-				
przykład:	JZ dalej				

LCALL adr_16	wywoływanie podprogramu (ang. long subroutine call)
---------------------	--

opis: Instrukcja bezwarunkowo wywołuje podprogram definiowany przez 16-bitowy adres bezpośredni.

bajty/cykle: 3/2

kodowanie:	<table border="1"><tr><td>0 0 0 1</td><td>0 0 0 1</td></tr></table>	0 0 0 1	0 0 0 1	<table border="1"><tr><td>a₁₅ a₁₄ a₁₃ a₁₂</td><td>a₁₁ a₁₀ a₉ a₈</td></tr></table>	a ₁₅ a ₁₄ a ₁₃ a ₁₂	a ₁₁ a ₁₀ a ₉ a ₈
0 0 0 1	0 0 0 1					
a ₁₅ a ₁₄ a ₁₃ a ₁₂	a ₁₁ a ₁₀ a ₉ a ₈					
		<table border="1"><tr><td>a₇ a₆ a₅ a₄</td><td>a₃ a₂ a₁ a₀</td></tr></table>	a ₇ a ₆ a ₅ a ₄	a ₃ a ₂ a ₁ a ₀		
a ₇ a ₆ a ₅ a ₄	a ₃ a ₂ a ₁ a ₀					

działanie: PC ← PC + 3
(SP) ← (SP) + 1
((SP)) ← (PC₇₋₀)
(SP) ← (SP) + 1

$$((SP)) \leftarrow (PC_{15-8})$$

$$(PC) \leftarrow \text{adr}_{16}$$

zmiana flag: -

przykład: Po uruchomieniu mikrokontrolera, SP jest ustawiany na wartość 07h. Jeżeli etykieta *podprogram* wskazuje na podprogram umieszczony pod adresem 1234h, to po wykonaniu instrukcji:

$$\text{ACALL } \text{podprogram}$$

umieszczonej pod adresem 0123h, licznik rozkazów, PC, przyjmie wartość 1234h a wskaźnik stosu, SP, wartość 09h. W pamięci RAM, pod adresami 08h i 09h będą umieszczone bajty o wartościach, odpowiednio, 26h i 01h.

LJMP *adr*₁₆

skok bezwarunkowy długi
(ang. long jump)

opis: Przekazuje wykonywanie programu pod adres określony słowem 16-bitowym.

bajty/cykle: 3/2

kodowanie:

0	0	0	0	0	0	1	0	a ₁₅	a ₁₄	a ₁₃	a ₁₂	a ₁₁	a ₁₀	a ₉	a ₈
								a ₇	a ₆	a ₅	a ₄	a ₃	a ₂	a ₁	a ₀

działanie: $(PC) \leftarrow \text{adr}_{16}$

zmiana flag: -

przykład: LJMP dalej

MOV <dest>, <src>

przesyłanie bajtu
(ang. move byte variable)

opis: Instrukcja odczytuje bajt spod adresu <src> i zapisuje go pod adres <dest>. Dana pod adresem <src> pozostaje bez zmian. Instrukcja MOV pozwala na 15 sposobów adresowania danych.

przykład: Jeżeli pamięć RAM pod adresem 30h zawiera bajt o wartości 40h a pod adresem 40h jest bajt o wartości 10h oraz gdy stan portu P1 jest określony stanem 11001010b (0CAh) to wykonanie grupy instrukcji:

$$\begin{array}{lll} \text{MOV} & \text{R0, \#30h} & ; \text{R0} \leftarrow 30\text{h} \\ \text{MOV} & \text{A, @R0} & ; \text{A} \leftarrow 40\text{h} \end{array}$$

```

MOV    R1, A        ; R1 ← 40h
MOV    B, @R1       ; B ← 10h
MOV    @R1, P1      ; RAM(40h) ← 0CAh
MOV    P2, P1       ; P2 ← #0CAh

```

wprowadza do akumulatora wartość 40h, do rejestru B wartość 10h, do rejestru R0 wartość 30h, do rejestru R1 wartość 40h, do portu P1 wartość 03Ah oraz do pamięci RAM, pod adres 40h, wartość 03Ah.

MOV A, Rn

przesyłanie bajtu

opis: Instrukcja kopiuje do akumulatora stan rejestru roboczego R0..R7.

bajty/cykle: 1/1

kodowanie:

1	1	1	0	1	r_0	r_1	r_2
---	---	---	---	---	-------	-------	-------

działanie: $(A) \leftarrow (Rn)$ gdzie $Rn \in \{R0, \dots, R7\}$

zmiana flag: -

przykład: MOV A, R5

MOV A, @Ri

przesyłanie bajtu

opis: Instrukcja kopiuje do akumulatora stan rejestru adresowanego pośrednio przez R0 lub R1.

bajty/cykle: 1/1

kodowanie:

1	1	1	0	1	1	1	i
---	---	---	---	---	---	---	-----

gdzie $i \in \{0,1\}$

działanie: $(A) \leftarrow ((Ri))$ gdzie $Rn \in \{R0, R1\}$

zmiana flag: -

przykład: MOV A, @R1

MOV A, adr

przesyłanie bajtu

opis: Instrukcja kopiuje do akumulatora stan rejestru adresowanego bezpośrednio.

bajty/cykle: 1/1

kodowanie:	<table border="1"><tr><td>1 1 1 0</td><td>0 1 0 1</td></tr></table>	1 1 1 0	0 1 0 1	<table border="1"><tr><td>adr</td></tr></table>	adr
1 1 1 0	0 1 0 1				
adr					
działanie:	(A) ← (adr)				
zmiana flag:	-				
przykład:	MOV A, 123 MOV A, adres				
uwagi:	Instrukcja: MOV A, ACC jest niedozwolona.				

MOV A, #n	przesyłanie bajtu
------------------	-------------------

opis:	Instrukcja wpisuje do akumulatora stałą n .				
bajty/cykle:	2/1				
kodowanie:	<table border="1"><tr><td>0 1 1 1</td><td>0 1 0 0</td></tr></table>	0 1 1 1	0 1 0 0	<table border="1"><tr><td>n</td></tr></table>	n
0 1 1 1	0 1 0 0				
n					
działanie:	(A) ← n				
zmiana flag:	-				
przykład:	MOV A, #123				

MOV Rn, A	przesyłanie bajtu
------------------	-------------------

opis:	Instrukcja kopiuje stan akumulatora do rejestru roboczego R0..R7.			
bajty/cykle:	1/1			
kodowanie:	<table border="1"><tr><td>1 1 1 1</td><td>1 r₀ r₁ r₂</td></tr></table>	1 1 1 1	1 r ₀ r ₁ r ₂	
1 1 1 1	1 r ₀ r ₁ r ₂			
działanie:	(Rn) ← (A)	gdzie $Rn \in \{R0, \dots, R7\}$		
zmiana flag:	-			
przykład:	MOV R5, A			

MOV Rn, adr	przesyłanie bajtu
--------------------	-------------------

opis:	Instrukcja kopiuje do rejestru R0..R7 stan rejestru adresowanego bezpośrednio.	
bajty/cykle:	2/2	

kodowanie:

1 0 1 0	1 r ₀ r ₁ r ₂
---------	--

adr

działanie: $(R_n) \leftarrow (\text{adr})$ gdzie $R_n \in \{R_0, \dots, R_7\}$
zmiana flag: -
przykład: MOV R5, 123

MOV Rn, #n	przesyłanie bajtu
-------------------	-------------------

opis: Instrukcja kopiuje do rejestru R0..R7 stałą n .
bajty/cykle: 2/1
kodowanie:

0 1 1 1	1 r ₀ r ₁ r ₂
---------	--

n

działanie: $(R_n) \leftarrow n$ gdzie $R_n \in \{R_0, \dots, R_7\}$
zmiana flag: -
przykład: MOV R5, #123

MOV @Ri, A	przesyłanie bajtu
-------------------	-------------------

opis: Instrukcja kopiuje stan akumulatora do rejestru adresowanego pośrednio przez R0 lub R1.
bajty/cykle: 1/1
kodowanie:

1 1 1 1	0 1 1 i
---------	---------

 gdzie $i \in \{0,1\}$
działanie: $((R_i)) \leftarrow (A)$ gdzie $R_i \in \{R_0, R_1\}$
zmiana flag: -
przykład: MOV @R1, A

MOV @Ri, adr	przesyłanie bajtu
---------------------	-------------------

opis: Instrukcja kopiuje stan rejestru adresowanego bezpośrednio do rejestru adresowanego pośrednio przez R0 lub R1.
bajty/cykle: 2/2
kodowanie:

1 0 1 0	0 1 1 i
---------	---------

 gdzie $i \in \{0,1\}$
działanie: $((R_i)) \leftarrow (\text{adr})$ gdzie $R_i \in \{R_0, R_1\}$
zmiana flag: -

przykład: MOV @R1, 123

MOV @Ri, #n

przesyłanie bajtu

opis: Instrukcja wpisuje stałą n do rejestru adresowanego pośrednio przez R0 lub R1.

bajty/cykle: 2/1

kodowanie:

0 1 1 1	0 1 1 i	n
---------	---------	---

działanie: ((Ri)) ← n

zmiana flag: -

przykład: MOV A, #123

MOV adr, A

przesyłanie bajtu

opis: Instrukcja kopiuje stan akumulatora do rejestru adresowanego bezpośrednio.

bajty/cykle: 2/1

kodowanie:

1 1 1 1	0 1 0 1	adr
---------	---------	-----

działanie: (adr) ← (A)

zmiana flag: -

przykład: MOV 123, A

MOV adr, Rn

przesyłanie bajtu

opis: Instrukcja kopiuje stan rejestru R0..R7 do rejestru adresowanego bezpośrednio.

bajty/cykle: 2/2

kodowanie:

1 0 0 0	0 1 0 0	adr
---------	---------	-----

działanie: (adr) ← (Rn)

zmiana flag: -

przykład: MOV 123, R5

MOV adr, @Ri

przesyłanie bajtu

opis: Instrukcja kopiuje stan rejestru adresowanego pośrednio przez R0 lub R1 do rejestru adresowanego bezpośrednio.

bajty/cykle: 2/2

kodowanie:

1	0	0	0
---	---	---	---

0	1	1	i
---	---	---	---

adr

 $i \in \{0,1\}$

działanie: $(adr) \leftarrow ((Ri))$ gdzie $Ri \in \{R0, R1\}$

zmiana flag: -

przykład: MOV 123, @R1

MOV adr_1, adr_2

przesyłanie bajtu

opis: Instrukcja kopiuje stan rejestru adresowanego bezpośrednio do innego rejestru, też adresowanego bezpośrednio.

bajty/cykle: 3/2

kodowanie:

1	0	0	0
---	---	---	---

0	1	0	1
---	---	---	---

adr_2

adr_1

działanie: $(adr_1) \leftarrow (adr_2)$

zmiana flag: -

przykład: MOV P1, ACC

MOV adr, #n

przesyłanie bajtu

opis: Instrukcja wpisuje stałą n do do rejestru adresowanego bezpośrednio.

bajty/cykle: 3/2

kodowanie:

1	0	0	0
---	---	---	---

0	1	0	1
---	---	---	---

adr

n

działanie: $(adr) \leftarrow n$

zmiana flag: -

przykład: MOV 123, #123

MOV DPTR, #nnprzesyłanie danej 16 bitowej
(ang. load data pointer with
16-bit constant)**opis:** Instrukcja wpisuje do rejestrów wskaźnika danych daną 16-bitową.**bajty/cykle:** 3/2**kodowanie:**

1 0 0 1 | 0 0 0 0

n_{DPH}n_{DPL}**działanie:** (DPTR) ← nn**zmiana flag:** -**przykład:** MOV DPTR, #1234h**MOV <dst_bit>, <src_bit>**przesyłanie bitu
(ang. move bit data)**opis:** Instrukcja kopiuje bit spod adresu <src_bit> do adresu <dst_bit>. Jednym z bitów musi być bit CY. Drugi bit musi być adresowany bezpośrednio. Bit pod adresem <src_bit> pozostaje bez zmian. Instrukcja przesyłania bitu obejmuje 2 sposoby adresowania argumentu.**przykład:** Jeżeli stan wejść portu P3 jest równy 0C5h (11000101b); wartość uprzednio wpisana do rejestru portu P1 wynosi 35h (00110101b) oraz bit CY ma wartość 1, to wykonanie instrukcji:

```
MOV P1.3, C
MOV C, P3.3
MOV P1.2, C
```

zmienia stan portu P1 na 39h (00111001b) i kasuje stan bitu CY (CY=0).

MOV bit, C

przesyłanie bitu

opis: Instrukcja kopiuje stan bitu CY do bitu o adresie bezpośrednim <bit>.**bajty/cykle:** 2/2

kodowanie:

1 0 0 1	0 0 1 0	bit
---------	---------	-----

działanie: (bit) ← (CY)

zmiana flag: -

przykład: MOV 123, C

MOV C, bit

przesyłanie bitu

opis: Instrukcja kopiuje do CY stan bitu o adresie bezpośrednim <bit>.

bajty/cykle: 2/1

kodowanie:

1 0 1 0	0 0 1 0	bit
---------	---------	-----

działanie: (CY) ← (bit)

zmiana flag: -

przykład: MOV C, 123

MOVC A, @A+<reg_16>przesyłanie bajtu z pamięci programu
(ang. move code byte)

opis: Instrukcja dodaje stan akumulatora do stanu 16-bitowego rejestru bazowego i spod tak stworzonego adresu pobiera bajt z pamięci programu. W momencie dodawania, stan akumulatora jest traktowany jako naturalna liczba binarna. Jest to jedyna instrukcja pozwalająca na odczyt dowolnej danej z pamięci programu przez program. Instrukcja MOVC obejmuje 2 sposoby adresowania argumentu.

MOVC A, @A+DPTR

przesyłanie bajtu z pamięci programu

opis: Instrukcja dodaje stan akumulatora do rejestru DPTR i spod tak stworzonego adresu pobiera bajt z pamięci programu. W momencie dodawania, stan akumulatora jest traktowany jako naturalna liczba binarna.

bajty/cykle: 1/2

kodowanie:

1 0 0 1	0 0 1 1
---------	---------

działanie: $(A) \leftarrow ((A) + (DPTR))$

zmiana flag: -

przykład: Jeżeli *offset* jest liczbą o wartości od 0 do 3, to wykonanie instrukcji:

```
MOV    A, offset
MOV    DPTR, #tabela
MOVC  A, @A + DPTR
```

.....

```
tabela: DB    55h
         DB    65h
         DB    75h
         DB    85h
```

spowoduje pobranie danej z tablicy *tabela* i wpisanie jej do akumulatora. Dana będzie pobrana z miejsca przesuniętego o wartość *offset* od początku tablicy.

MOVC A, @A+PC

przesyłanie bajtu z pamięci programu

opis: Instrukcja dodaje stan akumulatora do rejestru PC i spod tak stworzonego adresu pobiera bajt z pamięci programu. W momencie dodawania, stan akumulatora jest traktowany jako naturalna liczba binarna a stan rejestru PC jest równy adresowi następczej instrukcji po MOVC.

bajty/cykle: 1/2

kodowanie:

1 0 0 0	0 0 1 1
---------	---------

działanie: $(PC) \leftarrow (PC) + 1$
 $(A) \leftarrow ((A) + (PC))$

zmiana flag: -

przykład: Jeżeli *offset* jest liczbą o wartości od 0 do 3 i liczba ta jest umieszczona w akumulatorze, to wykonanie podprogramu *xxx*:

```
xxx:    INC    A
        MOVC  A, @A + PC
        RET
```

```
tabela: DB    55h
         DB    65h
         DB    75h
```

DB 85h

spowoduje pobranie danej z tablicy *tabela* i wpisanie jej do akumulatora. Dana będzie pobrana z miejsca przesuniętego o wartość *offset* od początku tablicy. Tabela danych musi być, w takim przypadku, umieszczona bezpośrednio za podprogramem.

MOVX <dest>, <src>	przesyłanie bajtu pomiędzy mikrokontrolerem a zewnętrzną pamięcią RAM (ang. move external)
opis:	Instrukcja zapisuje bajt danej do zewnętrznej RAM lub odczytuje bajt z tej pamięci. Instrukcja działa w dwu trybach. W pierwszym z nich, adresowanie zewnętrznej pamięci RAM jest 8 bitowe i informacja o adresie jest przekazywana za pośrednictwem portu P0. W drugim trybie, adresowanie zewnętrznej pamięci RAM jest 16 bitowe i informacja o adresie jest przekazywana za pośrednictwem portu P0 oraz P2 - przez port P0 jest przekazywana młodsza część adresu a przez P2 część starsza. W obu przypadkach, wysyłany jest dodatkowy sygnał sterowania: sygnał WR w przypadku zapisywania danej do pamięci lub sygnał RD w przypadku odczytywania danej. Instrukcja MOVX obejmuje 4 sposoby adresowania argumentu.
przykład:	W rejestrze R0 jest dana o wartości 12h a w rejestrze R1 - 34h. Jeżeli w zewnętrznej pamięci RAM, mającej rozmiar 256 bajtów, pod adresem 34h jest bajt o wartości 55h, to wykonanie instrukcji: <pre>MOVX A, @R1 MOVX @R0, A</pre> spowoduje skopiowanie danej o wartości 55h do akumulatora i do pamięci RAM pod adres 12h.

MOVX A, @ DPTR	przesyłanie bajtu z zewnętrznej pamięci RAM
-----------------------	---

opis: Instrukcja kopiuje do akumulatora bajt z zewnętrznej pamięci RAM adresowanej bezpośrednio przez rejestr DPTR. Adresowanie zewnętrznej pamięci RAM jest 16 bitowe.

bajty/cykle: 1/2

kodowanie:

1 1 1 0	0 0 0 0
---------	---------

działanie: $(A) \leftarrow ((DPTR))$

zmiana flag: -

przykład: `MOVX A, @DPTR`

MOVX A, @ Ri

przesyłanie bajtu z zewnętrznej pamięci RAM

opis: Instrukcja kopiuje do akumulatora bajt z zewnętrznej pamięci RAM adresowanej pośrednio przez rejestr R0 lub R1. Adresowanie zewnętrznej pamięci RAM jest 8 bitowe.

bajty/cykle: 1/2

kodowanie:

1 1 1 0	0 0 1 i
---------	---------

gdzie $i \in \{0,1\}$

działanie: $(A) \leftarrow ((Ri))$

gdzie $Ri \in \{R0, R1\}$

zmiana flag: -

przykład: `MOVX A, @R0`

MOVX @ DPTR, A

przesyłanie bajtu do zewnętrznej pamięci RAM

opis: Instrukcja kopiuje stan akumulatora do zewnętrznej pamięci RAM adresowanej bezpośrednio przez rejestr DPTR. Adresowanie zewnętrznej pamięci RAM jest 16 bitowe.

bajty/cykle: 1/2

kodowanie:

1 1 1 1	0 0 0 0
---------	---------

działanie: $((DPTR)) \leftarrow (A)$

zmiana flag: -

przykład: `MOVX @DPTR, A`

MOVX @ Ri, A

przesyłanie bajtu do zewnętrznej pamięci RAM

opis: Instrukcja kopiuje stan akumulatora do zewnętrznej pamięci RAM adresowanej pośrednio przez rejestr R0 lub R1. Adresowanie zewnętrznej pamięci RAM jest 8 bitowe.

bajty/cykle: 1/2

kodowanie:

1	1	1	1	0	0	1	i
---	---	---	---	---	---	---	---

gdzie $i \in \{0,1\}$

działanie: $((Ri)) \leftarrow (A)$

gdzie $Ri \in \{R0, R1\}$

zmiana flag: -

przykład: MOVX @R0, A

MUL AB

mnożenie
(ang. multiply)

opis: Instrukcja wykonuje operację mnożenia dwu liczb 8-bitowych bez znaku - zawartość akumulatora jest mnożona przez stan rejestru B. Starsza część 16 bitowego wyniku jest wpisywana do rejestru B a młodsza część do akumulatora. Instrukcja kasuje flagę CY. Flaga OV jest ustawiana w przypadku, gdy wynik mnożenia jest większy od 255; w każdym innym przypadku flaga OV jest kasowana.

uwaga: Jeżeli stan rejestru B jest zerem (00h) to po wykonaniu instrukcji stan akumulatora i rejestru B są nieokreślone - flaga CY jest kasowana a flaga OV ustawiana w stan jedynki logicznej.

bajty/cykle: 1/4

kodowanie:

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

działanie: (B) \leftarrow starsza część z $[(A)*(B)]$

(A) \leftarrow młodsza część z $[(A)*(B)]$

zmiana flag: CY=0, OV=0 lub CY=0, OV=1 gdy wynik mnożenia > 255

przykład: Jeżeli stan akumulatora jest równy 80 (50h) a stan rejestru B wynosi 160 (0A0h) to po wykonaniu operacji mnożenia:

MUL A, B

w akumulatorze znajdzie się liczba 0 (00h) a w rejestrze B liczba 50 (32h) ponieważ: $50*256 + 0 = 12800 (=80*160)$

NOP A

instrukcja "pusta"
(ang. no operation)

opis: Instrukcja nie wykonuje żadnej operacji - jest używana do wprowadzania opóźnienia w wykonywaniu programu o 1 cykl maszynowy.

bajty/cykle: 1/1

kodowanie:

0 0 0 0	0 0 0 0
---------	---------

działanie: $(PC) \leftarrow (PC) + 1$

zmiana flag: -

ORL <dest >, <src >	sumowanie logiczne bajtów (ang. logical OR for byte variables)
<p>opis: Instrukcja wykonuje operację sumy logicznej OR na odpowiadających sobie bitach dwu bajtów, umieszczonych pod adresami <dest> i <src>. Wynik operacji jest umieszczany pod adresem <dest>. Instrukcja sumy logicznej bajtów obejmuje 6 sposobów adresowania argumentu.</p> <p>przykład: Jeżeli akumulator zawiera daną o wartości 0C3h (11000011b) a w rejestrze R0 jest dana o wartości 55h (01010101b) to wykonanie instrukcji:</p> <p style="text-align: center;">ORL A, R0</p> <p>wprowadza do akumulatora wartość 0D7h (110101111b).</p> <p>uwaga: Jeżeli instrukcja jest użyta do modyfikacji stanu portów P0..P3, daną do wykonania instrukcji jest stan rejestrów linii portu a nie stan końcówek portu. Wynik operacji jest zapisywany do rejestru portu i może wpływać na stan końcówek portu.</p>	

ORL A, Rn	suma logiczna bajtów		
<p>opis: Instrukcja wykonuje operację OR pomiędzy bitami akumulatora a bitami rejestru roboczego R0..R7.</p> <p>bajty/cykle: 1/1</p> <p>kodowanie: <table border="1" style="display: inline-table; border-collapse: collapse;"><tr><td style="padding: 2px 10px;">0 1 0 0</td><td style="padding: 2px 10px;">1 r₀ r₁ r₂</td></tr></table></p> <p>działanie: $(A) \leftarrow (A) \vee (Rn)$ gdzie $Rn \subset \{R0, .., R7\}$</p> <p>zmiana flag: -</p> <p>przykład: ORL A, R5</p>		0 1 0 0	1 r ₀ r ₁ r ₂
0 1 0 0	1 r ₀ r ₁ r ₂		

ORL A, adr

suma logiczna bajtów

opis: Instrukcja wykonuje operację OR pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą *adr*.

bajty/cykle: 2/1

kodowanie:

0 1 0 0	0 1 0 1	
---------	---------	--

adr

działanie: $(A) \leftarrow (A) \vee (\text{adr})$

zmiana flag: -

przykład: ORL A, adres
ORL A, 123

ORL A, @Ri

suma logiczna bajtów

opis: Instrukcja wykonuje operację OR pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1.

bajty/cykle: 1/1

kodowanie:

0 1 0 0	0 1 1 i	
---------	---------	--

 gdzie $i \in \{0, 1\}$

działanie: $(A) \leftarrow (A) \vee ((Ri))$ gdzie $Ri \in \{R0, R1\}$

zmiana flag: -

przykład: ORL A, @R1

ORL A, #n

suma logiczna bajtów

opis: Instrukcja wykonuje operację OR pomiędzy bitami akumulatora a bitami bajtu o wartości *n*.

bajty/cykle: 2/1

kodowanie:

0 1 0 0	0 1 0 0	
---------	---------	--

n

działanie: $(A) \leftarrow (A) \wedge n$

zmiana flag: -

przykład: ORL A, #123

ORL adr, A	suma logiczna bajtów
--------------------------	----------------------

opis: Instrukcja wykonuje operację OR pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą *adr*. Wynik operacji jest umieszczany pod adresem *adr*.

bajty/cykle: 2/1

kodowanie:

0 1 0 0	0 0 1 0
---------	---------

adr

działanie: $(adr) \leftarrow (adr) \vee (A)$

zmiana flag: -

przykład: ORL *adr*, A
ORL 123, A

ORL adr, #n	suma logiczna bajtów
---------------------------	----------------------

opis: Instrukcja wykonuje operację OR pomiędzy bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą *adr* a bitami bajtu o wartości *n*. Wynik operacji jest umieszczany pod adresem *adr*.

bajty/cykle: 3/2

kodowanie:

0 1 0 0	0 0 1 1
---------	---------

adr

n

działanie: $(adr) \leftarrow (adr) \vee n$

zmiana flag: -

przykład: ORL *adr*, #123
ORL 123, #123

ORL C, <src_bit>	sumowanie logiczne bitów (ang. logical OR for bit variables)
--------------------------------------	---

opis: Instrukcja wykonuje operację sumy logicznej OR pomiędzy bitem CY i bitem źródła, wskazywanego adresem bezpośrednim. Znak "/", poprzedzający adres bitu źródła oznacza, że instrukcja wykona operację sumy na negacji bitu źródła. Instrukcja sumy logicznej bitów obejmuje 2 sposoby adresowania bitu argumentu.

przykład: W celu określenia, czy spełniony jest warunek:

P1.0=1, ACC.7=1 oraz OV=0
 trzeba wykonać 3 instrukcje kodu:
 MOV C, P1.0
 ORL C, ACC.7
 ORL C, /OV
 jeżeli bit CY=1 to wyżej określony warunek jest spełniony.

ORL C, bit

suma logiczna bitów

opis: Instrukcja wykonuje operację OR pomiędzy bitem CY i bitem źródła.

bajty/cykle: 2/2

kodowanie:

0 1 1 1	0 0 1 0
---------	---------

adres bitu

działanie: $(CY) \leftarrow (CY) \vee (\text{bit})$

zmiana flag: CY

przykład: ORL C, TF0
 ORL C, 08Dh

ORL C, /bit

suma logiczna bitów

opis: Instrukcja wykonuje operację OR pomiędzy bitem CY i negacją bitu źródła.

bajty/cykle: 2/2

kodowanie:

1 0 1 0	0 0 0 0
---------	---------

adres bitu

działanie: $(CY) \leftarrow (CY) \vee /(\text{bit})$

zmiana flag: CY

przykład: ORL C, /TF0
 ORL C, /08Dh

POP adrpobranie ze stosu
(ang. pop from stack)

opis: Instrukcja wykonuje operację pobierania danej z wewnętrznej pamięci RAM spod adresu wskazywanego przez rejestr SP (odczytywanie z wierzchołka stosu). Pobrana dana jest umieszczana w rejestrze wskazywanym adresem

bezpośrednim. Po skopiowaniu danej, stan wskaźnika SP jest zmniejszany o 1.

bajty/cykle: 2/2

kodowanie:

1	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

działanie: (adr) \leftarrow ((SP))
(SP) \leftarrow (SP) - 1

zmiana flag: -

przykład: POP P1

PUSH adr

ładowanie na stos
(ang. push onto stack)

opis: Instrukcja wykonuje operację kopiowania danej wskazywanej adresem bezpośrednim do wewnętrznej pamięci RAM. Dana jest zapisywana pod adres wskazywany przez rejestr SP (zapisywanie na wierzchołek stosu). Przed wykonaniem kopiowania, stan wskaźnika SP jest zwiększany o 1.

bajty/cykle: 2/2

kodowanie:

1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

działanie: (SP) \leftarrow (SP) + 1
((SP)) \leftarrow (adr)

zmiana flag: -

przykład: PUSH P1

RET

powrót z podprogramu
(ang. return from subroutine)

opis: Instrukcja pobiera ze stosu 2 bajty 16-bitowego adresu i wprowadza je do rejestru PC - jest wykonywany skok bezwarunkowy. Wskaźnik stosu SP jest zmniejszany o 2.

bajty/cykle: 1/2

kodowanie:

0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

działanie: (PC₁₅₋₈) \leftarrow ((SP))
(SP) \leftarrow (SP) - 1
(PC₇₋₀) \leftarrow ((SP))
(SP) \leftarrow (SP) - 1

zmiana flag: -

RETIpowrót z przerwania
(ang. return from interrupt)

opis: Instrukcja pobiera ze stosu 2 bajty 16-bitowego adresu i wprowadza je do rejestru PC - jest wykonywany skok bezwarunkowy. Wskaźnik stosu SP jest zmniejszany o 2. Instrukcja jest rozpoznawana przez kontroler przerwania.

bajty/cykle: 1/2

kodowanie:

0 0 1 1	0 0 1 0
---------	---------

działanie: $(PC_{15-8}) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) - 1$
 $(PC_{7-0}) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) - 1$

zmiana flag: -

RL Aprzesuwanie bitów akumulatora
w lewo
(ang. rotate accumulator left)

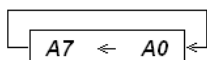
opis: Instrukcja wykonuje operację przesunięcia wszystkich bitów akumulatora o jedną pozycję w lewo. Bit z pozycji najstarszej jest przesuwany w pozycję najmłodszą.

bajty/cykle: 1/1

kodowanie:

0 0 1 0	0 0 1 1
---------	---------

działanie: $(A_n + 1) \leftarrow (A_n)$
 $(A_0) \leftarrow (A_7)$



zmiana flag: -

RLC Aprzesuwanie bitów akumulatora
i bitu CY w lewo
(ang. rotate accumulator left
through the carry flag)

opis: Instrukcja wykonuje operację przesunięcia wszystkich bitów akumulatora oraz bitu CY o jedną pozycję w lewo. Bit CY jest przesuwany w pozycję najmłodszą akumulatora. Bit z pozycji najstarszej akumulatora jest przesuwany do CY.

bajty/cykle: 1/1

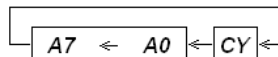
kodowanie:

0 0 1 1	0 0 1 1
---------	---------

działanie: $(A_n + 1) \leftarrow (A_n)$ gdy $n < 7$

$(A_0) \leftarrow (CY)$

$(CY) \leftarrow (A_7)$



zmiana flag: CY

RR A

przesuwanie bitów akumulatora
w prawo
(ang. rotate accumulator right)

opis: Instrukcja wykonuje operację przesunięcia wszystkich bitów akumulatora o jedną pozycję w prawo. Bit z pozycji najmłodszej jest przesuwany w pozycję najstarszą.

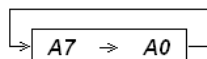
bajty/cykle: 1/1

kodowanie:

0 0 0 0	0 0 1 1
---------	---------

działanie: $(A_n) \leftarrow (A_n + 1)$

$(A_7) \leftarrow (A_0)$



zmiana flag: -

RRC A

przesuwanie bitów akumulatora
i bitu CY w prawo
(ang. rotate accumulator right
through the carry flag)

opis: Instrukcja wykonuje operację przesunięcia wszystkich bitów akumulatora oraz bitu CY o jedną pozycję w prawo. Bit CY jest przesuwany w pozycję najstarszą akumulatora. Bit z pozycji najmłodszej akumulatora jest przesuwany do CY.

bajty/cykle: 1/1

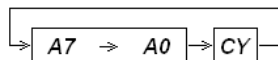
kodowanie:

0 0 0 1	0 0 1 1
---------	---------

działanie: $(A_n) \leftarrow (A_n + 1)$ gdy $n < 7$

$(A_7) \leftarrow (CY)$

$(CY) \leftarrow (A_0)$



zmiana flag: CY

SETB bit

ustawianie bitu
(ang. set bit)

opis: Instrukcja wykonuje operację ustawiania bitu wskazywanego adresem bezpośrednim.

bajty/cykle: 1/1

kodowanie:

1 1 0 1	0 0 1 0
---------	---------

działanie: (bit) \leftarrow 1

zmiana flag: -

przykład: Jeżeli stan rejestru portu P1 był równy 5Dh (01011101b), to po wykonaniu operacji:

SETB P1.1

będzie on wynosił 5Fh (10101111b).

SETB C

ustawianie bitu

opis: Instrukcja wykonuje operację ustawiania bitu CY.

bajty/cykle: 1/1

kodowanie:

1 1 0 1	0 0 1 1
---------	---------

działanie: (CY) \leftarrow 1

zmiana flag: CY

SJMP d

skok względny bezwarunkowy
(ang. short jump)

opis: Instrukcja wykonuje operację skoku względnego bezwarunkowego. Adres docelowy jest tworzony przez dodanie do licznika rozkazów PC przesunięcia d - wykonywany jest skok względny. Przesunięcie d jest liczbą ze znakiem, zapisaną w kodzie U2. Skok jest wykonywany względem adresu następnej instrukcji po SJMP.

bajty/cykle: 2/2

kodowanie:

1 0 0 0	0 0 0 0	d
---------	---------	---

działanie: (PC) \leftarrow (PC + 2)

(PC) \leftarrow (PC) + d

zmiana flag: -

przykład: SJMP dalej

SUBB A, <src>

wykonywanie odejmowania
(ang. subtract with borrow)

opis: Od zawartości akumulatora odejmowany jest wskazany bajt oraz bit CY (pożyczka). Wynik operacji jest umieszczany w akumulatorze. Operacja odejmowania wpływa na stan znaczników CY, AC i OV.

Flaga CY jest ustawiana, gdy bitowi A_7 jest potrzebna pożyczka i kasowana gdy pożyczka jest zbędna (jeżeli CY jest ustawione przed wykonaniem instrukcji SUBB, oznacza to, że pożyczka jest niezbędna w pierwszym kroku wykonywania odejmowania liczb 16 bitowych i jest ona odejmowana od akumulatora wspólnie z odjemnikiem).

Flaga AC jest ustawiana, gdy bitowi A_3 jest potrzebna pożyczka i kasowana gdy pożyczka jest zbędna.

Flaga OV jest ustawiana, gdy pożyczka jest potrzebna bitowi A_6 i niepotrzebna bitowi A_7 lub na odwrót.

W przypadku odejmowania liczb ze znakiem, ustawiona flaga OV wskazuje na wynik ujemny odejmowania w przypadku odejmowania liczby ujemnej od dodatniej. Flaga jest ustawiana gdy od liczby ujemnej jest odejmowana liczba dodatnia a wynik odejmowania jest dodatni.

Instrukcja SUBB obejmuje 4 sposoby adresowania argumentu.

przykład: Jeżeli akumulator zawiera daną o wartości 0C9h (11001001b) a w rejestrze R2 jest dana o wartości 0AAh (10101010b) to instrukcja:

```
SUBB A,R2
```

wprowadzi do akumulatora wartość 74h (01110100b) - flagi CY i AC będą wykasowane a flaga OV będzie ustawiona.

SUBB A, Rn

odejmowanie

opis: Od zawartości akumulatora odejmowany jest bit CY i stan rejestru roboczego R0..R7.

bajty/cykle: 1/1

kodowanie:

1	0	0	1
---	---	---	---

1	r ₀	r ₁	r ₂
---	----------------	----------------	----------------

działanie: $(A) \leftarrow (A) - (CY) - (Rn)$ gdzie $Rn \in \{R0, \dots, R7\}$

zmiana flag: CY, AC i OV

przykład: SUBB A, R5

SUBB A, adr

odejmowanie

opis: Od zawartości akumulatora odejmowany jest bit CY i stan rejestru umieszczonego pod adresem bezpośrednim, oznaczonym etykietą: *adr*.

bajty/cykle: 2/1

kodowanie:

1	0	0	1
---	---	---	---

0	1	0	1
---	---	---	---

adr

działanie: $(A) \leftarrow (A) - (CY) - (adr)$

zmiana flag: CY, AC i OV

przykład: SUBB A, adres
SUBB A, 123

SUBB A, @Ri

odejmowanie

opis: Od zawartości akumulatora odejmowany jest bit CY i stan rejestru umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1.

bajty/cykle: 1/1

kodowanie:

1	0	0	1
---	---	---	---

0	1	1	i
---	---	---	---

 gdzie $i \in \{0,1\}$

działanie: $(A) \leftarrow (A) - (CY) - ((Ri))$ gdzie $Ri \in \{R0, R1\}$

zmiana flag: CY, AC i OV

przykład: SUBB A, @R1

SUBB A, #n

odejmowanie

- opis:** Od zawartości akumulatora odejmowany jest bit CY i bajt o wartości n .
- bajty/cykle:** 2/1
- kodowanie:**

1 0 0 1	0 1 0 0	n
---------	---------	---
- działanie:** $(A) \leftarrow (A) - (CY) - n$
- zmiana flag:** CY, AC i OV
- przykład:** SUBB A, #123

SWAP A

zamiana półbajtów miejscami
(ang. swap nibbles within the accumulator)

- opis:** Instrukcja wykonuje operację wymiany młodszej czwórki bitów akumulatora z czwórką starszą akumulatora.
- bajty/cykle:** 1/1
- kodowanie:**

1 1 0 0	0 1 0 0
---------	---------
- działanie:** $(A_{3..0}) \leftrightarrow ((A)_{7..4})$
- | | |
|--------|--------|
| A7..A4 | A3..A0 |
|--------|--------|
- zmiana flag:** -
- przykład:** Jeżeli akumulator zawiera daną o wartości 36h (00110110b) to wykonanie instrukcji:
SWAP A
stan akumulatora będzie równy 63h (01100011b).

XCH A, <src >

wymiana zawartości akumulatora i bajtu pamięci wewnętrznej
(ang. exchange accumulator with byte variable)

- opis:** Instrukcja wykonuje operację wymiany danych pomiędzy akumulatorem a bajtem umieszczonym pod adresem <src>. Instrukcja obejmuje 3 sposoby adresowania argumentu.
- przykład:** Jeżeli w wewnętrznej pamięci RAM, pod adresem 55h jest dana o wartości 0Fh; jeżeli akumulator zawiera daną o wartości 0C3h a w rejestrze R0 jest dana o wartości 55h to wykonanie instrukcji:

XCHD A, @Ri

wymiana półbajtów pomiędzy akumulatorem a bajtem pamięci (ang. exchange digit)

opis: Instrukcja wykonuje operację wymiany młodszej czwórki bitów akumulatora z młodszą czwórką bitów bajtu pamięci wewnętrznej, adresowanego pośrednio rejestrami R0 lub R1.

bajty/cykle: 1/1

kodowanie:

1	1	0	1	0	1	1	i
---	---	---	---	---	---	---	---

 gdzie $i \in \{0, 1\}$

działanie: $(A_{3..0}) \Leftrightarrow ((Ri)_{3..0})$

zmiana flag: -

przykład: Jeżeli akumulator zawiera daną o wartości 36h (00110110b) a w rejestrze adresowanym pośrednio przez R0 jest dana o wartości 75h (01110101b) to wykonanie instrukcji:

XCHD A, @R0

wprowadza do akumulatora wartość 35h (00110101b) a do wskazywanego przez R0 wartość 76h (01110110b).

XRL <dest >, <src >

sumowanie logiczne bitów bajtu w trybie modulo 2 (ang. logical ExOR for byte variables)

opis: Instrukcja wykonuje operację sumy modulo 2 na odpowiadających sobie bitach dwu bajtów, umieszczonych pod adresami <dest> i <src>. Wynik operacji jest umieszczany pod adresem <dest>.

Instrukcja XRL obejmuje 6 sposobów adresowania argumentu.

przykład: Jeżeli akumulator zawiera daną o wartości 0C3h (11000011b) a w rejestrze R0 jest dana o wartości AAh (10101010b) to wykonanie instrukcji:

XRL A,R0

wprowadza do akumulatora wartość 69h (01101001b).

uwaga: Jeżeli instrukcja jest użyta do modyfikacji stanu portów P0..P3, daną do wykonania instrukcji jest stan rejestrów linii portu a nie stan końcówek portu.

XRL A, Rn

suma modulo 2

opis: Instrukcja wykonuje operację sumy modulo 2 pomiędzy bitami akumulatora a bitami rejestru roboczego R0..R7.

bajty/cykle: 1/1

kodowanie:

0	1	1	0
---	---	---	---

1	r ₀	r ₁	r ₂
---	----------------	----------------	----------------

działanie: $(A) \leftarrow (A) \oplus (R_n)$ gdzie $R_n \subset \{R_0, \dots, R_7\}$

zmiana flag: -

przykład: XRL A, R5

XRL A, adr

suma modulo 2

opis: Instrukcja wykonuje operację sumy modulo 2 pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą *adr*.

bajty/cykle: 2/1

kodowanie:

0	1	1	0
---	---	---	---

0	1	0	1
---	---	---	---

adr

działanie: $(A) \leftarrow (A) \oplus (\text{adr})$

zmiana flag: -

przykład: XRL A, adres
XRL A, 123

XRL A, @Ri

suma modulo 2

opis: Instrukcja wykonuje operację sumy modulo 2 pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem pośrednio wskazywanym przez rejestry R0 lub R1.

bajty/cykle: 1/1

kodowanie:

0	1	1	0
---	---	---	---

0	1	1	i
---	---	---	---

 gdzie $i \in \{0, 1\}$

działanie: $(A) \leftarrow (A) \oplus ((R_i))$ gdzie $R_i \subset \{R_0, R_1\}$

zmiana flag: -

przykład: XRL A, @R1

XRL A, #n

suma modulo 2

opis: Instrukcja wykonuje operację sumy modulo 2 pomiędzy bitami akumulatora a bitami bajtu o wartości n .

bajty/cykle: 2/1

kodowanie:

0 1 1 0	0 1 0 0	n
---------	---------	---

działanie: $(A) \leftarrow (A) \oplus n$

zmiana flag: -

przykład: XRL A, #123

XRL adr, A

suma modulo 2

opis: Instrukcja wykonuje operację sumy modulo 2 pomiędzy bitami akumulatora a bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą adr . Wynik operacji jest umieszczany pod adresem adr .

bajty/cykle: 2/1

kodowanie:

0 1 1 0	0 0 1 0	adr
---------	---------	-----

działanie: $(adr) \leftarrow (adr) \oplus (A)$

zmiana flag: -

przykład: XRL adr, A
XRL 123, A

XRL adr, #n

suma logiczna bajtów

opis: Instrukcja wykonuje operację sumy modulo 2 pomiędzy bitami rejestru umieszczonego pod adresem bezpośrednim, opisanym etykietą adr a bitami bajtu o wartości n . Wynik operacji jest umieszczany pod adresem adr .

bajty/cykle: 3/2

kodowanie:

0 1 1 0	0 0 1 1	adr	n
---------	---------	-----	---

działanie: $(adr) \leftarrow (adr) \oplus n$

zmiana flag: -

przykład: XRL adr, #123
XRL 123, #123

BIBLIOGRAFIA

- 1 "8-bit Embedded Controller Handbook"; tom 3 z 11-tomowego katalogu produktów f-my Intel Corporation, 1990, (nr katalogowy: 270645);
- 2 "Apache Developers C Language Style Guide";
<http://httpd.apache.org/dev/styleguide.html>
- 3 Gałka Piotr, Gałka Paweł : "Podstawy programowania mikrokontrolera 8051", MIKOM, Warszawa 2006 (wydanie IV), ISBN 978-23-01-14789-1;
- 4 Heinz W.W.: "MCS-51 Microcontroller Family Macro Assembler - User Manual"; dokumentacja pakietu ASEM-5, zawarta w pliku "asem_51.doc", 2002;
- 5 instrukcja obsługi: "DSM-51, Dydaktyczny System Mikroprocesorowy"; dokumentacja systemu DSM_51 zawarta w pliku "dsm51_IO.pdf", Micro-Made, 2007r;
- 6 Owczarek Tomasz: "Laboratorium podstaw techniki mikroprocesorowej i elementów konstrukcji systemów cyfrowych"; Oficyna Wydawnicza Politechniki Warszawskiej; Warszawa 1999, ISBN 83-7207-133-0;
- 7 Rydzewski Andrzej: "Mikrokomputery jednokładowe rodziny MCS-51"; WNT, Warszawa, 1992; ISBN 83-204-1401-6;
- 8 Sacha Krzysztof , Rydzewski Andrzej, "Mikroprocesor w pytaniach i odpowiedziach"; WNT, Warszawa 1985, ISBN 83-204-0684-6;
- 9 "SDCC Compiler User Guide, SDCC 2.9.0, 2009-03-13, Revision: 5413";
sdcc.sourceforge.net/doc/sdccman.pdf
- 10 opis produktu: "AT24C01A/02/04/08/16, 2-Wire Serial CMOS E2PROM"; Atmel Corporation, 1999.
- 11 opis produktu: "8-bit Microcontroller with 8K Bytes In-System Programmable Flash AT89S8253"; Atmel Corporation, 2008;
- 12 opis produktu: "ADuC841/ADuC842/ADuC843 MicroConverter® 12-Bit ADCs and DACs with Embedded High Speed 62-kB Flash MCU"; Analog Devices, 2003;
- 13 opis produktu: "Enhanced 8-bit Microcontroller with 32 KB Flash Memory, AT89C51AC2"; Atmel Corporation, 2008;
- 14 opis produktu: "HD44780U (LCD-II) - Dot Matrix Liquid Crystal Display Controller/Driver"; Hitachi, Ltd., 1998;
- 15 opis produktu: "MAX220-MAX249, +5V-Powered, Multichannel RS-232 Drivers/Receivers"; Maxim Integrated Products, 1997;

- 16 opis standardu: "The I²C Bus Specification, Version 2.1"; Philips Semiconductors; January 2000; document order number: 9398 393 40011;
- 17 Starecki Tomasz: "mikrokontrolery jednoukładowe rodziny 51"; wydawnictwo "NOZOMI", Warszawa, 1996, ISBN 83-906175-0-1;
- 18 Wójciak Andrzej, "Mikroprocesory w energo-elektronice"; WNT, Warszawa 1984, ISBN 83-204-0591-2;

SKOROWIDZ

A

adres

bazowy..... 70, 71, 73, 79
bezpośredni..... 165, 188
fizyczny 88
pośredni167, 169, 171, 180, 183,
 202, 210, 214
powrotu..... 135, 136
procedury przerwaniowej..... 136
wewnętrzny..... 88, 90

adresowanie

bezpośrednie..... 13, 14
bitowe 37, 38, 39, 50, 51, 55, 56, 58,
 60
pośrednie..... 14, 15, 16, 159

ASCII..... 74, 75, 121, 124, 129

ASEM-51..... 121

assembler

patrz: język programowania

ASEMW: 121

B

bank

pamięci 84
RB0..... 15
RB1..... 15, 84
RB2..... 15
RB3..... 15
rejestrów..... 14, 15, 136, 151

bit

flagowy28, 29, 30, 32, 33, 37, 38,
 39, 47, 51, 68, 124, 135
kontrolny ..26, 30, 32, 33, 35, 36, 37,
 38, 39, 40, 50, 52, 56, 60, 145,
 147, 158
startu 45, 46, 47
stopu45, 47, 49, 50, 51, 63, 106,
 109, 111, 114

bit kontrolny

AC77, 78, 145, 147, 151, 159,
 164, 166, 167, 168, 169, 178, 179,
 209, 210, 211
C/T26, 27, 29, 30, 31, 33, 36, 38,
 40, 145, 148, 154, 155, 156

C/T230, 31, 33, 36, 38, 40, 145,
 148, 155, 156

CP/RL2 ..30, 31, 32, 33, 38, 40, 145,
 148, 155, 156

CY16, 145, 147, 151, 159, 160,
 162, 166, 167, 168, 169, 173, 174,
 175, 176, 177, 178, 179, 180, 181,
 185, 186, 187, 195, 196, 200, 203,
 204, 206, 207, 208, 209, 210, 211

DCEN..... 35, 36, 39, 40, 158

DISRTO..... 60, 61, 158

DPS 58, 158

EA8, 19, 55, 131, 132, 145, 147,
 149, 152, 164

ES47, 55, 131, 132, 145, 147,
 149, 152

ET055, 131, 132, 145, 147, 149,
 152

ET1 55, 145, 147, 149, 152

ET2 55, 145, 147, 149, 152

EX0..... 55, 145, 147, 149, 152

EX1..... 55, 145, 147, 149, 152

EXEN230, 32, 33, 35, 38, 145,
 148, 155

EXF230, 32, 33, 34, 37, 39, 40,
 145, 148, 155

F0 145, 147, 151, 164

GATE.. 26, 27, 29, 38, 145, 148, 154

GF0 51, 145, 147, 150

GF1 51, 145, 147, 150

HWDT..... 60, 61, 158

IDL 12, 51, 145, 147, 150

IE0..... 37, 56, 145, 148, 153

IE1 37, 56, 145, 148, 153

IT0 37, 56, 145, 148, 153

IT1 37, 56, 145, 148, 153

M0 26, 38, 145, 148, 154

M1 26, 38, 41, 43, 145, 148, 154

OV145, 147, 151, 159, 166, 167,
 168, 169, 173, 178, 180, 181, 200,
 204, 209, 210, 211

P 151

PD 12, 51, 146, 147, 150

PS 56, 146, 147, 149

PS0 59, 60, 61, 158

- PS1 59, 61, 158
 PS2 59, 61, 158
 PT0 56, 146, 147, 149
 PT1 56, 146, 147, 149
 PT2 56, 146, 147, 149
 PX0 56, 144, 146, 147, 149
 PX1 56, 144, 146, 147, 149
 RB8 45, 47, 49, 50, 51, 146, 147,
 152
 RCLK 30, 31, 32, 33, 34, 38, 39,
 40, 146, 148, 155, 156
 REN 41, 49, 51, 146, 147, 152
 RI 41, 43, 44, 47, 50, 51, 82, 83,
 131, 134, 146, 147, 152
 RS0 15, 84, 146, 147, 151
 RS1 15, 84, 146, 147, 151
 SM0 41, 50, 146, 147, 152
 SM1 41, 50, 146, 147, 152
 SM2 41, 49, 50, 146, 147, 152
 SMOD 45, 46, 47, 48, 51, 146, 147,
 150
 T2OE 35, 39, 40, 158
 TB8 45, 46, 47, 51, 146, 147, 152
 TCLK 30, 31, 32, 33, 34, 38, 39,
 40, 146, 148, 155, 156
 TF0 28, 30, 37, 56, 146, 148, 153,
 173, 174, 204
 TF1 28, 29, 37, 56, 146, 148, 153
 TF2 30, 32, 33, 34, 35, 37, 39, 40,
 146, 148, 155
 TI 41, 43, 44, 47, 51, 82, 83, 84,
 108, 109, 131, 134, 140, 146, 147,
 152
 TR0 29, 37, 56, 131, 132, 146,
 148, 153
 TR1 29, 37, 56, 131, 132, 146,
 148, 153, 154
 TR2 30, 32, 33, 38, 39, 146, 148,
 155, 156
 WDIDLE 60, 61, 158
 WDTEN 61, 62, 158
 WSWRST 61, 62, 158
 bit kontrolny HD44780
 B 77
 BF 78
 C 77
 D 77
 DL 76, 77, 78
 F 78
 I/D 76, 77, 78
 N 77
 R/L 76, 77
 S 77
 S/C 76, 77
C
 C *patrz: język programowania*
 CG_RAM 75, 76, 77, 78, 79
 czas krytyczny 113, 114
 czasomierz .. 25, 26, 28, 47, 48, 68, 132
 częstotliwość 8, 35, 36, 41, 44, 45,
 46, 47, 51, 69, 95, 150
D
 DD_RAM 75, 76, 77
 dostęp
 bezpośredni 66, 69
 pośredni 66
 sekwencyjny 69
 DSM-51
 patrz: system mikroprocesorowy
 dyrektywa
 #define 131, 134, 139
 #include 135
 DB 123
 END 123, 126
 EQU 123, 124
 ORG 123, 124, 125, 217
E
 EEPROM 5, 6, 57, 58, 86, 88, 89,
 90, 114
 etykieta 16, 122, 165, 170, 189
F
 flaga 28, 32, 33, 34, 56, 57, 69, 72,
 74, 80, 85, 86, 91, 93, 96, 101, 102,
 104, 105, 107, 108, 109, 116, 117,
 119, 124, 126, 131, 132, 134, 140,
 145, 146, 147, 148, 151, 152, 153,
 155, 159, 166, 168, 178, 180, 200,
 209
 FTSM_51
 patrz: system mikroprocesorowy
G
 generator
 cyklu zegarowego 69

dźwięku 103
 fali prostokątnej 35, 39
 mikrokontrolera 8, 27
 sygnału taktowania 34
 sygnału zegarowego 5, 29, 31, 47
 wewnętrzny 31, 36
 zdarzeń 1, 68
 GNU 128

I

I2C patrz magistrala I²C
 I²C: 85, 86, 87, 88, 90, 91, 114, 218
 instrukcja
 adresowania 15, 25, 35, 37
 asemblera 123
 dostępu 13
 for 130
 mikrokontrolera 15, 19, 122, 123
 RMW 16, 20
 skoku 19, 136
 sterująca 76
 switch 128, 130
 while 130, 132, 139
 własna 123

J

język programowania
 asembler 99, 104, 108, 116, 121,
 122, 124
 C 99, 127, 129, 130, 131, 134,
 135, 137, 138, 140
 niskiego poziomu 121
 wysokiego poziomu 121, 128

K

klawiatura 65, 71, 111, 112, 113
 kod
 wynikowy 123, 129
 źródłowy 121, 122, 123, 124, 128,
 129, 130, 135
 komentarz 122, 123
 kontroler przerwań 51, 52, 57, 118, 206
 kontrolka
 \$INCLUDE 123
 \$LIST 123
 \$NOLIST 123
 \$NOMOD51 123

L

LED 68, 72, 73, 74, 96, 97, 124,
 126, 131, 132, 133, 138, 139
 Linux 128

Ł

łącze
 I²C 85, 86, 87, 88, 90, 91, 114
 RS232 1, 63, 64, 67, 81
 SPI 6, 57
 USB 67

M

magistrala
 adresowa 3, 9, 144
 danych 3, 77, 144
 dwukierunkowa 9, 23
 I²C 86, 88
 jednokierunkowa 9
 mikroprocesora 9
 sterująca 3
 mapa 18, 70
 mapowanie 131, 134
 mikrokontroler 4, 6, 9, 18, 19, 40, 44,
 49, 50, 59, 61, 63, 75, 86, 88, 89, 90
 mikroprocesor 2, 3, 5, 16, 52, 57, 85,
 100, 101, 103, 118
 mnemonik 122, 160, 161, 162, 163
 moduł 41, 73, 78, 97

O

obsługa
 enkodera 91, 93
 klawiatury 68, 69, 97
 łącza I²C 85, 90, 91
 modułu 79
 odbioru 83
 pętli programowej 80, 90
 portu szeregowego 81, 82, 85
 przerwania 15, 52, 55, 57, 84,
 118, 119, 126, 133
 silnika 93, 95, 96
 wyświetlacza 68, 72, 73, 74, 79,
 80, 81, 97, 139
 zdarzenia 27, 28, 32, 69, 82, 83,
 84, 101, 102, 104, 105, 106, 107,
 108, 109, 112, 113, 114, 115, 116,

- 117, 118, 119, 120, 121, 123, 126,
127, 140
- zewnętrzna..... 67
- zintegrowana..... 67
- odczytywanie
- bajtu..... 15, 89, 114
- bitu..... 15, 20, 47
- danej10, 11, 13, 14, 19, 70, 76,
78, 88, 89, 198
- informacji..... 10, 14, 17, 23, 90
- klawiatury..... 70, 71, 137, 138
- kodu..... 8, 10
- licznika 26, 28, 109, 117
- pliku..... 111
- portu..... 12
- statusu..... 76, 78
- P**
- pamięć
- CG_RAM..... 75, 76, 77, 78, 79
- DD_RAM..... 75, 76, 77
- EEPROM5, 6, 57, 58, 86, 88,
89, 90, 114
- programu 2, 6, 8, 11, 13, 18, 19,
52, 57, 74, 79, 82, 84, 102, 122,
123, 126, 127, 138, 139, 140, 162,
165, 170, 196, 197
- RAM2, 5, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 20, 25, 64, 66, 68,
69, 72, 74, 75, 77, 79, 80, 81, 82,
83, 84, 85, 86, 90, 91, 93, 96, 101,
103, 112, 124, 136, 139, 140, 142,
159, 165, 189, 190, 198, 199, 204,
205, 211, 212
- ROM2, 5, 9, 10, 18, 19, 63, 64,
75, 79, 80, 81
- wewnętrzna ..5, 9, 12, 13, 16, 18, 19,
24, 68, 74, 79, 82, 84, 204, 205,
211, 213
- zewnętrzna8, 10, 11, 12, 18, 19,
21, 23, 137, 139, 140, 198, 199
- pętla
- pętla w pętli..... 106, 107, 109, 120
- pomocnicza. 105, 106, 107, 109, 110
- programowa59, 69, 72, 74, 81,
82, 84, 85, 91, 93, 96, 97, 98, 99,
102, 103, 104, 105, 106, 109, 113,
115, 117, 119, 120, 121, 126, 127,
135, 136, 140
- pusta..... 98, 105, 106, 112
- plik
- nagłówkowy..... 131, 135
- wynikowy..... 129
- podprogram59, 68, 69, 73, 74, 75,
79, 81, 82, 83, 84, 85, 92, 93, 94, 95,
96, 98, 123, 127, 165, 188, 189
- pole
- adresowe..... 70, 76
- banków 14
- bitowe 14, 15, 159
- danych 13, 14, 17
- magnetyczne 95
- okna menu 66
- okna nastaw 19, 66, 92, 94, 159, 160
- okna pomocy..... 67
- pamięci13, 14, 17, 18, 74, 82, 84,
85, 91, 112, 159
- pamięci CG_RAM..... 77, 78, 143
- pamięci danych13, 14, 17, 20, 84,
160
- pamięci DD_RAM75, 76, 77, 78,
79, 143
- pamięci programu18, 19, 74, 159,
160
- pamięci RAM14, 17, 68, 74, 79,
80, 81, 82, 83, 84, 85, 86, 91, 112,
124, 151, 159
- pulpitu..... 63
- SFR13, 14, 16, 17, 20, 24, 25,
157, 159
- stykowe 92
- tekstowe 64, 75
- wskaznika 75
- wyświetlacza..... 75, 76, 77, 78, 79
- znakowe 75
- port
- P06, 7, 9, 10, 12, 17, 20, 21, 22,
23, 25, 157, 170, 178, 179, 181,
182, 185, 198, 201, 213
- P17, 9, 16, 17, 20, 21, 22, 23,
24, 31, 32, 33, 34, 35, 36, 38, 39,
66, 132, 144, 155, 156, 157, 173,
177, 180, 181, 183, 184, 185, 189,
190, 194, 195, 204, 205, 208
- P27, 9, 10, 11, 12, 16, 17, 21,
22, 23, 24, 33, 35, 157, 190, 198
- P35, 6, 7, 16, 17, 20, 21, 22, 23,
24, 25, 40, 42, 43, 44, 47, 66, 81,

- 144, 157, 170, 178, 179, 181, 182,
185, 195, 201, 213
- szeregowy ...5, 40, 43, 44, 48, 51, 52,
55, 81, 82, 83, 85, 105, 107, 108,
110, 124, 125, 126, 133, 134, 135,
136
- uniwersalny 1, 5, 7, 9, 20, 21, 23, 24
- preprocesor 134, 135
- priorytet 18, 52, 53, 54, 57, 137, 149
- procedura
- obsługi wyświetlacza* 97
- opóźniająca* 110
- przerwaniowa* 82, 83
- program
- awaryjny* 120
- długi* 113, 114, 115, 116
- krótki* 113, 114
- obsługi przerwania*15, 18, 19, 32,
 51, 52, 57, 82, 118, 119, 121
- podstawowy* 120
- przerwanie28, 29, 30, 35, 40, 47, 51,
52, 53, 54, 55, 69, 84, 118, 119, 124,
125, 126, 131, 132, 133, 134, 135,
136, 140, 144, 149, 150, 152
- przygotowanie środowiska 103, 120
- R**
- RAM2, 5, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 20, 25, 64, 66, 68, 69, 72,
74, 75, 77, 79, 80, 81, 82, 83, 84, 85,
86, 90, 91, 93, 96, 101, 103, 112,
124, 136, 139, 140, 142, 159, 165,
189, 190, 198, 199, 204, 205, 211,
212
- rejestr
- ACC16, 17, 20, 80, 84, 91, 95,
 137, 157, 159, 173, 174, 187, 191,
 194, 204
- adresowy* 77
- aktywacji przerwania* 149
- buforowy* 41
- danych* 76, 79, 103
- DPH 10, 12, 16, 17, 195
- DPL 10, 16, 17, 91, 195
- DPTR6, 10, 16, 20, 57, 58, 80,
 84, 137, 159, 160, 161, 162, 163,
 164, 165, 184, 186, 195, 196, 197,
 198, 199
- kontrolny*26, 37, 38, 39, 47, 49,
 52, 53, 55, 56, 58, 60, 78, 79, 100,
 102, 125, 132, 147, 149, 158
- licznika* 16, 26, 28, 30, 124, 127
- mikrokontrolera* 16, 84
- mikroprocesora* 103
- modułu LCD* 76
- nadajnika* 41, 43
- odbiornika* 41
- ogólnego przeznaczenia* 15
- pamięci* 16, 30, 72, 75, 76, 93, 96
- PC10, 16, 20, 65, 67, 81, 136,
 159, 160, 162, 163, 164, 165, 170,
 174, 175, 176, 181, 182, 184, 185,
 186, 187, 188, 189, 197, 201, 205,
 206, 208, 209
- PCH 10
- PCL 10
- PISO 41, 43
- pomocniczy*30, 31, 32, 33, 35, 60,
 80, 91
- portu* 20, 25
- priorytetu przerwania* 149
- przesuwany* 41, 42, 43, 47
- PSW15, 16, 17, 84, 136, 145, 146,
 147, 151, 157
- R010, 14, 15, 84, 151, 159, 163,
 164, 166, 167, 168, 169, 170, 171,
 175, 176, 179, 180, 181, 182, 183,
 184, 189, 190, 191, 192, 193, 194,
 198, 199, 200, 201, 202, 210, 211,
 212, 213, 214
- R110, 14, 15, 84, 159, 163, 164,
 167, 169, 171, 172, 176, 180, 183,
 184, 190, 192, 193, 194, 198, 199,
 200, 202, 210, 212, 213, 214
- R2 15, 84, 163, 164, 181, 209
- R3 84, 163, 164
- R4 163, 164
- R5163, 164, 167, 168, 171, 176,
 180, 182, 183, 190, 191, 192, 193,
 201, 210, 212, 214
- R6 163, 164
- R715, 84, 151, 159, 163, 164,
 166, 168, 171, 174, 175, 179, 180,
 181, 182, 183, 190, 191, 192, 193,
 201, 210, 212, 214
- RCAP 30, 31, 32, 33, 35, 36, 37, 155
- RCAPH 30

- RCAPL*..... 30
roboczy 15, 84
SBUF17, 41, 42, 43, 44, 45, 46,
 47, 49, 50, 82, 83, 105, 131, 133,
 134, 140, 157
SIPO 41, 42, 43, 44, 47
SP16, 17, 103, 157, 159, 162,
 163, 165, 188, 189, 204, 205, 206
stanu 16
statusu liczników..... 153
statusu programu..... 151
statusu zasilania 150
sterowania licznikiem..... 155
sterowania portem szeregowym.. 152
strefy SFR..... 16, 41, 60, 140
TH0... 17, 26, 29, 131, 132, 134, 157
TH1..... 17, 48, 131, 132, 154, 157
TH2..... 17, 30, 157
TL017, 26, 29, 131, 132, 134,
 154, 157
TL1 17, 26, 131, 132, 154, 157
TL2 17, 30, 157
trybu pracy liczników 154
urządzenia 101, 103
zatrząskowy 9
 rejestr kontrolny
EECON..... 58, 157, 158
IE17, 47, 52, 53, 55, 57, 145,
 147, 149, 152, 157
IP 17, 47, 53, 54, 55, 146, 147,
 149, 150, 157
PCON ..12, 17, 45, 47, 51, 131, 132,
 145, 146, 147, 150, 157
SCON17, 41, 43, 45, 49, 50, 82,
 131, 132, 146, 147, 152, 157
T2CON17, 30, 38, 39, 145, 146,
 148, 155, 157
T2MOD..... 35, 36, 39, 157, 158
TCON17, 26, 37, 56, 145, 146,
 148, 153, 154, 157
TMOD17, 26, 37, 131, 132, 145,
 148, 154, 157, 175
WDTCN 59, 60, 157, 158
ROM2, 5, 9, 10, 18, 19, 63, 64, 75,
 79, 80, 81
RS232: 48, 49, 65, 66, 67, 81, 144
- S**
SDCC128, 129, 132, 136, 137, 138,
 140, 217
SFR13, 16, 26, 30, 35, 37, 50, 58,
 123, 124, 134, 159
 słowo kluczowe
 __code 138, 140
 __interrupt..... 134, 136
 __using 136
 __xdata..... 140
char129, 130, 132, 133, 137, 138,
 139
int 129, 130
main..... 132, 139
 stan
 aktywny9, 10, 11, 28, 38, 56, 57,
 60, 69, 86, 101, 102, 105, 107,
 108, 155
 pasywny38, 39, 56, 60, 86, 101,
 102, 103, 107
stos15, 17, 18, 124, 125, 136, 162,
 205
 sygnał
 ALE..... 7, 9, 10
 alternatywny..... 23
 bramkowania..... 38
 CS 76
 EA8, 19, 55, 131, 132, 145, 147,
 149, 152, 164
 fosc8, 41, 43, 45, 46, 47, 48, 59,
 61, 76, 154, 156
 generatora..... 34, 38, 39, 59
 kasowania..... 25, 52, 53, 59, 97
 odbierany..... 41, 45, 47
 odczytu..... 144
 OSC 8, 27, 31, 34
 periodyczny..... 35
 przepiętnia 45, 46
 przerwania.... 24, 35, 51, 52, 57, 153
 przesuwający 41, 42
 PSEN 8, 10
 R/W..... 76, 88, 90
 RCLK30, 31, 32, 33, 34, 38, 39,
 40, 146, 148, 155, 156
 RD 8, 10, 12, 23, 24, 144, 198
 reset 7, 57
 RSHIFT..... 42, 43, 44, 47
 RST 7, 19, 25, 60, 61, 158

RXC 46
RxD 24, 67, 81
RXT 34
S1PI 10
S3PI 12, 43
S4PI 10
SCL 86, 87, 88, 90
SDA 86, 87, 90
ST 20, 47
START 88, 132
sterowania 9, 21, 198
STOP 88
synchronizacji 44
taktowania 24, 26, 27, 29, 30, 31,
36, 38, 41, 42, 45, 46, 48, 155,
156
TCLK 30, 31, 32, 33, 34, 38, 39,
40, 146, 148, 155, 156
TCR 43, 45
TCT 43, 45
TD 41, 47
TSHIFT 41, 43
TXC 34, 46, 47
TxD 24, 67, 81
wewnętrzny 27, 29, 31, 40, 154, 156
WG_EN 59
WG_RES 59
WR 10, 12, 23, 24, 144, 198
wyzwalania 24
XTAL1 8
XTAL2 8
zapisu 144
zegarowy 8, 35, 44
zewnątrzny 27, 29, 31, 32, 36, 38, 40
system mikroprocesorowy
DSM-51 63, 64, 67, 68, 75, 98,
126, 137, 217
FTSM_51 58, 63, 65, 67, 70, 73,
75, 79, 90, 92, 94, 95, 96, 141,
144
szybkość transmisji 36, 41, 43, 46,
47, 48, 109, 150

Ś

środowisko 4, 6, 37, 59, 67, 68, 98,
99, 100, 101, 102, 103, 104, 105,
106, 107, 108, 109, 110, 111, 113,
114, 115, 116, 117, 118, 120, 121,
126, 127, 135

T

testowanie stanu środowiska 104
transmisja
asynchroniczna 40, 45, 63, 152
danych 78
synchroniczna 40, 43, 152
tryb
adresowania 10
asynchroniczny 44, 45, 81, 105, 114
autoprzetadowania 30, 31
autoregeneracji 28
awaryjny 120
DSM-51 63
DSMX 63, 64
dwukierunkowy 9, 86
dwuliniowy 77
energooszczędny 12
funkcji alternatywnych 21
generatora szybkości transmisji
..... 30, 33, 35
generatora zegarowego 35
komplementarny 23
licznika rewersyjnego 36, 39
mikroprocesorowy 7, 8, 9, 21, 23
obniżonego poboru mocy 12, 51
pasywny 67
portu szeregowego 41
portu uniwersalnego 23, 24
pracy alternatywnej 23, 24
pracy 'awaryjnej' 120
pracy jałowej 12, 51, 150
pracy 'normalnej' 120
programowy 61
przechwytywania 30, 32, 33, 38
standardowy 39, 120
sterowania sprzętowego 61, 62
synchroniczny 43, 44
tryb 0 26, 40, 43, 50
tryb 1 26, 38, 44, 45, 46, 47, 48,
50, 51, 68, 150, 155
tryb 2 28, 29, 50, 125, 132
tryb 3 29, 50
wejściowy 7, 23
wirtualny 95
wyjściowy 7, 22, 23, 24
zwiększonej szybkości 58

U

urządzenie

<i>I/O</i>	3, 10, 11, 12, 16, 23, 24, 51, 52, 57, 68, 70, 72, 73, 74, 75, 81, 82, 85, 86, 91, 92, 93, 94, 96, 102, 103, 119, 144
<i>mikroprocesorowe</i>	12
<i>nadrzędne</i>	86, 87
<i>otoczenia</i>	85
<i>peryferyjne</i>	61, 65, 86
<i>podrzędne</i>	85, 86
<i>systemu</i>	3, 85
<i>wewnętrzne</i>	6
<i>wirtualne</i>	63
<i>zewnętrzne</i>	9, 18, 19, 21, 24, 52, 53, 54, 57
USB	66, 67

W

<i>watchdog</i>	6, 59, 60, 97
WDT	57, 58, 59, 60, 61, 62, 97, 98, 158
wektor	
<i>przerwania</i>	18, 52, 53
Windows	128
wskaźnik	
<i>adresowy</i>	84, 89, 90
<i>danych</i>	6

<i>DPTR</i>	58, 158, 186
<i>LCD</i>	79, 103
<i>LED</i>	65, 74, 97
<i>stosu</i>	16, 17, 103, 137, 159, 165, 189

wyświetlacz

<i>dynamiczny</i>	72, 73
<i>graficzny</i>	74
<i>LCD</i>	64, 65, 74, 75, 76, 81
<i>LED</i>	72, 73, 97, 138, 139, 140
<i>multipleksowany</i>	72, 73, 74
<i>sekwencyjny</i>	72
<i>statyczny</i>	72
<i>tekstowy</i>	74, 79

Z

zapisywanie

<i>bajtu</i>	15, 112
<i>bezpośrednie</i>	20
<i>bitu</i>	15
<i>danej</i>	10, 11, 12, 13, 14, 17, 26, 30, 75, 76, 80, 88, 89, 198
<i>informacji</i>	14, 17, 23
zdarzenia powiązane	110, 111, 112, 113, 121, 127
złącze	
CANNON	81
IDC 40	65, 66, 144
zmienna globalna	136, 137, 140