

Mikrokontrolery 8051 *w praktyce*

Tomasz Starecki

0755 880-4000

89L58252 89C52 89C54 89L58252 89C55

btc

Mikrokontrolery '51 pomimo swojej długiej historii (Intel wprowadził je na rynek w 1980 roku) praktycznie się nie zestarzały. Dostępność szerokiej gamy narzędzi wspomagających realizację projektów na '51, możliwość zakupu wielu wersji i niskie ceny tych mikrokontrolerów, bogate wyposażenie w bloki peryferyjne, a także ogromna liczba bezpłatnych gotowych procedur powodują, że rodziną '51 interesują się coraz to nowe pokolenia elektroników.

Książka „Mikrokontrolery '51 w praktyce” jest podręcznikiem zawierającym kompleksowy opis listy rozkazów i architektury mikrokontrolerów pochodnych 8051, który uzupełniono prezentacją procedur assemblerowych najczęściej stosowanych w aplikacjach użytkowników. Są to m.in. obsługa klawiatur, sterowanie wyświetlaczami LED i LCD, obsługa interfejsów RS232, SPI i I-Wire, a także programowa implementacja zegara czasu rzeczywistego.

Książka jest przeznaczona dla studentów wydziałów elektroniki i automatyki, inżynierów-projektantów wykorzystujących w swoich projektach mikrokontrolery z rdzeniem 8051, a także wszystkich miłośników techniki mikroprocesorowej, którzy zamierzają poznać 8051 od strony praktycznej.

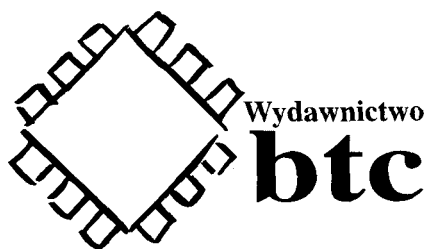
Recenzent: dr inż. Wojciech Zabłotny

Skład komputerowy: Marek Włodarz

Rysunek: Piotr Kanarek, www.kanarek.art.pl

ISBN 83-910067-4-3

© Copyright by Wydawnictwo BTC
Warszawa 2002.



Wydawnictwo BTC
ul. Inowłodzka 5
03-237 Warszawa
fax: (22) 782-42-90
<http://www.btc.pl>
e-mail: redakcja@btc.pl

Wydanie 1. Warszawa 2002.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawnictwo BTC dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawnictwo BTC nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentów niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Druk i oprawa: Łódzka Drukarnia Dziełowa S.A.

Spis treści

Od autora	6
1. Wprowadzenie	7
1.1. Mikroprocesor a mikrokontroler	8
1.2. Zastosowania mikrokontrolerów	8
1.3. Elementy typowego systemu mikroprocesorowego	9
2. Architektura mikrokontrolerów rodziny '51	11
2.1. Organizacja pamięci	12
2.1.1. Pamięć programu	12
2.1.2. Wewnętrzna pamięć danych	13
2.1.3. Rejestry ogólnego zastosowania	14
2.1.4. Stos	15
2.1.5. Zewnętrzna pamięć danych	15
2.1.6. Rejestry specjalnego przeznaczenia (SFR)	16
2.2. Taktowanie CPU	17
2.2.1. Generator sygnału zegarowego	17
2.2.2. Taktowanie mikrokontrolera	18
2.3. Zerowanie mikrokontrolera (RESET)	20
2.4. Budowa portów	23
2.5. Współpraca mikrokontrolera z układami zewnętrznymi	27
2.5.1. Komunikacja z zewnętrznymi układami peryferyjnymi	27
2.5.2. Instrukcje typu odczyt-modyfikacja-zapis	31
2.5.3. Obciążalność portów	32
2.5.4. Taktowanie zewnętrznych urządzeń peryferyjnych	33
3. Wewnętrzne układy peryferyjne mikrokontrolerów rodziny '51	35
3.1. Układy licznikowe T0 i T1	36
3.2. Układ licznikowy T2	39
3.2.1. Licznik T2 mikrokontrolera 8052	39
3.2.2. Modyfikacje układu licznikowego T2	41
3.3. Interfejsy szeregowo	43
3.3.1. Standardowe łącze szeregowe	43
3.3.2. Programowanie prędkości transmisji interfejsu	47
3.3.3. Przesyłanie informacji łączem szeregowym w systemach wieloprocessorowych	48
3.3.4. Rozszerzone łącze szeregowe	49
3.3.5. Interfejs SPI	50
3.4. Komparatory analogowe	53
3.5. Pamięć danych typu EEPROM	53
3.5.1. Pamięć danych typu EEPROM w mikrokontrolerze S8252	54
3.6. Układy czuwające	55
3.6.1. Liczniki czuwające	55
3.6.2. Układy monitorowania napięcia zasilającego	56

4.	Układ przerwań	57
4.1.	Struktura układu przerwań	58
4.2.	Obsługa przerwań	61
4.3.	Czas reakcji na przerwanie	62
4.4.	Przerwania zewnętrzne	63
4.4.1.	Przerwania $\overline{\text{INT0}}$ i INT1	63
5.	Zmniejszanie poboru mocy mikrokontrolerów	65
5.1.	Tryb uśpienia (<i>Idle</i>)	66
5.2.	Tryb zamrożenia (<i>Power Down</i>)	67
5.3.	Inne możliwości zmniejszania poboru mocy mikrokontrolerów	69
6.	Wewnętrzna pamięć programu	73
6.1.	Pamięć programu typu EPROM	74
6.2.	Pamięć programu typu Flash	78
6.2.1.	Programowanie pamięci Flash mikrokontrolerów o zmniejszonej liczbie wyprowadzeń	78
6.2.2.	Programowanie pamięci Flash pozostałych mikrokontrolerów firmy Atmel w odmianach C i LV	81
6.2.3.	Programowanie równoległe pamięci Flash mikrokontrolerów 89S51, S52, S53, S8252	82
6.2.4.	Programowanie szeregowo pamięci Flash mikrokontrolerów 89S51, S52, S53, S8252	84
6.2.5.	Pamięć Flash w wybranych mikrokontrolerach innych firm	87
7.	Lista instrukcji mikrokontrolerów rodziny '51	89
7.1.	Tryby adresowania	90
7.2.	Przegląd instrukcji	91
7.3.	Skrócona lista instrukcji mikrokontrolerów rodziny '51	138
8.	Rejestry SFR	143
8.1.	Rozmieszczenie rejestrów SFR	144
8.2.	Opis rejestrów SFR	155
9.	Wybrane narzędzia uruchomieniowe	175
9.1.	Asemblery	176
9.2.	Kompilatory języków wysokiego poziomu	177
9.3.	Kompilatory niestandardowe	180
9.4.	Symulatory	183
9.5.	Emulatory mikrokontrolerów	183
9.6.	Emulatory pamięci EPROM	184
9.7.	Monitory	184
9.8.	Debugery wewnętrzne	184
9.9.	Zestawy demonstracyjne, edukacyjne i prototypowe	187
9.10.	Programatory	187
9.10.1.	Programatory stacjonarne	188

9.10.2.	Programator wewnątrzkładowy firmy AEC Electronics.....	188
9.10.3.	Easy Downloader - prosty programator układów 89C2051 i C4051	191
9.10.4.	Oprogramowanie IC-Prog	192

10. Podstawy projektowania urządzeń z mikrokontrolerami rodziny '51 193

10.1.	Dołączenie do mikrokontrolera zewnętrznej pamięci programu	195
10.2.	Współpraca mikrokontrolera z zewnętrzną pamięcią danych i zewnętrznymi układami peryferyjnymi.....	196
10.3.	Obsługa klawiatur.....	196
10.4.	Sterowanie wyświetlacza i diod LED	201
10.5.	Współpraca mikrokontrolera z alfanumerycznym wyświetlaczem LCD	205
10.6.	Interfejs RS232	215
10.7.	Sterowanie układami wyposażonymi w interfejs SPI.....	219
10.8.	Interfejs I ² C	222
10.8.1.	Ogólne założenia interfejsu I ² C	223
10.8.2.	Przesyłanie informacji po szynie I ² C	224
10.8.3.	Mechanizm arbitrażu i synchronizacji	226
10.8.4.	Protokoły transmisji danych z adresowaniem 7-bitowym	228
10.8.5.	Protokoły transmisji danych z adresowaniem 10-bitowym	231
10.8.6.	Elektryczne parametry szyny I ² C	234
10.8.7.	Programowa implementacja interfejsu I ² C	237
10.9.	Interfejs 1-Wire.....	239
10.10.	Zegar czasu rzeczywistego	243
10.11.	Sterowanie odbiornikami zasilanymi napięciem sieciowym	244

11. Zestaw edukacyjny ZL1MCS51 245

12. Literatura i linki internetowe 261

Dodatki i uzupełnienia..... 265

Dodatek A.	Przegląd mikrokontrolerów rodziny '51	266
	Podstawowe parametry mikrokontrolerów rodziny '51	266
Dodatek B.	Dokumentacje techniczne	276
	B.1. Płytkę drukowaną programatora Easy Downloader.....	276
	B.2. Płytkę drukowaną zestawu ZL1MCS51	277
Dodatek C.	Wyprowadzenia typowych alfanumerycznych wyświetlaczy LCD	279
Dodatek D.	Rozmieszczenie wyprowadzeń wybranych typów mikrokontrolerów	280
Dodatek E.	Tablice kodów ASCII.....	283
Dodatek F.	Schematy blokowe wybranych mikrokontrolerów rodziny '51.....	287

Od autora

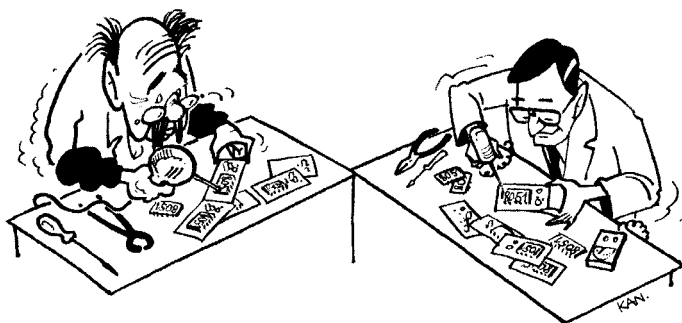
Nie ulega wątpliwości, że elektronika należy obecnie do jednej z najszybciej rozwijających się dziedzin techniki. Postęp ten jest bardzo wyraźny zwłaszcza w zakresie technik komputerowych – wystarczy uzmysłowić sobie, że w ciągu ostatnich 10 lat moc obliczeniowa komputerów osobistych, czy pojemność twardych dysków stosowanych w takim sprzęcie wzrosła co najmniej 100-krotnie. Rozwój technologii owocuje przy tym nie tylko znaczną poprawą parametrów, ale także bardzo szybkim spadkiem cen produkowanych urządzeń i poszczególnych ich podzespołów – w szczególności układów scalonych. Nierzadko okazuje się, że w porównaniu ze swoimi poprzednikami, nowsze opracowania charakteryzują się nie tylko lepszymi parametrami, ale także niższą ceną.

Bardzo szybki rozwój branży komputerowej ma też z pewnością znaczący wpływ na postęp w dziedzinie mikrokontrolerów jednoukładowych. Jednakże, o ile w przypadku komputerów zasadnicze znaczenie ma moc obliczeniowa stosowanych w nich procesorów i rozmiar pamięci, to w przypadku mikrokontrolerów, z racji innego rodzaju ich zastosowań, cechy te mają już znacznie mniejsze znaczenie – ważniejsze od dużej mocy obliczeniowej są zwykle możliwości wewnętrznych układów peryferyjnych. Z tego też powodu tanie mikrokontrolery 8-bitowe, a wśród nich najbardziej chyba rozpowszechnione w Europie i na polskim rynku mikrokontrolery rodziny '51, przeżywają w dalszym ciągu dynamiczny rozwój – w skład rodziny '51 wchodzi obecnie co najmniej kilkadziesiąt różnych mikrokontrolerów i co roku liczba ta wzrasta o kilkanaście nowych układów.

Sukces poprzedniej książki „Mikrokontrolery jednoukładowe rodziny 51” wykazał, że tematyka mikroprocesorowa, a w szczególności projektowanie systemów mikroprocesorowych z wykorzystaniem bardzo popularnych w naszym kraju układów rodziny '51, budzi cały czas szerokie zainteresowanie wśród Czytelników. Okazało się przy tym, że wiele osób szuka książki, w której oprócz wiadomości traktujących o najbardziej popularnych mikrokontrolerach znajdą się również informacje dotyczące podstawowych technik projektowania urządzeń (zarówno sprzętu, jak i oprogramowania) realizowanych z wykorzystaniem mikrokontrolerów. Oddawana do rąk Czytelnika książka miała być odpowiedzią na takie właśnie zapotrzebowanie.

Tomasz Starecki

Warszawa, 17.12.2002



Wprowadzenie



1.1. Mikroprocesor a mikrokontroler

Mikroprocesor jest wprawdzie niezbędnym, ale tylko jednym z elementów składowych większego systemu. Innymi ważnymi elementami systemu budowanego z wykorzystaniem mikroprocesora są układy wejścia/wyjścia, pamięć programu i pamięć danych. Układy wejścia/wyjścia umożliwiają komunikację systemu z otoczeniem, a pamięć programu i danych pozwala na przechowywanie kodu programu i informacji poddawanych obróbce. Wszystkie elementy takiego systemu mikroprocesorowego są ze sobą połączone za pomocą tzw. magistral (w szczególności magistrali adresowej i magistrali danych), czyli zespołów połączeń służących do przesyłania sygnałów niosących informację niezbędną do pracy całego systemu.

Mianem mikrokontrolera określa się pojedynczy układ scalony, którego struktura umożliwia samodzielne wykonywanie operacji arytmetycznych i sterowanie układami lub elementami zewnętrznymi. Mikrokontroler jest zatem układem scalonym zawierającym kompletny system, czyli CPU (centralną jednostkę procesora), pamięć i układy wejścia/wyjścia. Zaletami stosowania mikrokontrolerów są zatem m.in.: wysoka niezawodność uzyskiwana ze względu na niewielką liczbę elementów i połączeń zewnętrznych, krótki czas opracowywania, montażu i uruchamiania nowej konstrukcji.

Kompletny system, bez względu na to czy jest zrealizowany na mikroprocesorze, czy na mikrokontrolerze, nazywany jest systemem mikroprocesorowym.

1.2. Zastosowania mikrokontrolerów

To, co niegdyś było domeną układów logicznych małej i średniej skali integracji i programowanych układów logicznych, obecnie staje się obszarem zastosowań mikrokontrolerów. Olbrzymie znaczenie ma w tym przypadku niski koszt (kilkadziesiąt centów za sztukę w przypadku prostszych mikrokontrolerów) i olbrzymia elastyczność układu osiągana dzięki oprogramowaniu – między innymi także i dlatego, że oprogramowanie to może w pewnym zakresie zmieniać strukturę mikrokontrolera (widzianą z zewnątrz układu). Współczesne mikrokontrolery wymagają stosowania niewielu elementów zewnętrznych i są szeroko stosowane w urządzeniach powszechnego użytku – samochodach, pralkach, faksach, odbiornikach telewizyjnych, kuchenkach mikrofalowych itp. Z drugiej strony olbrzymie możliwości zastosowań mikrokontrolerów występują w przemyśle, gdzie mikrokontroler może być z równie dobrym skutkiem wykorzystany jako „serce” przyrządu pomiarowego, prostego sterownika, czy też urządzenia sterującego pracą całej linii technologicznej.

Na decyzję o użyciu mikrokontrolera w konkretnym zastosowaniu i wybór typu mikrokontrolera największy wpływ mają zazwyczaj dwa czynniki – parametry techniczne i koszt urządzenia. Parametry techniczne projektowanego urządzenia decydują z reguły o wyborze rodzaju mikrokontrolera. Wstępny wybór typu mikrokontrolera dokonywany jest najczęściej na drodze rozstrzygnięcia kolejno następujących problemów:

- czym ma sterować projektowane urządzenie,
- jakie wymagania muszą spełniać układy (urządzenia) wejścia/wyjścia,
- czego wymaga się od oprogramowania,
- jakiej szybkości pracy wymaga się od mikrokontrolera,
- jaki mikrokontroler będzie najlepszy dla danego zastosowania.

Przy porównywaniu kosztów wykonania urządzenia z zastosowaniem mikrokontrolera z kosztami tego samego urządzenia wykonanego inną techniką (np. z wykorzystaniem

konwencjonalnych układów logicznych lub układów PLD) należy brać oczywiście pod uwagę także i te koszty, jakie wynikają z opracowania układu, jego uruchamiania podczas produkcji, możliwości ewentualnych późniejszych zmian i modernizacji układu, niezawodności urządzenia itp.

1.3. Elementy typowego systemu mikroprocesorowego

Typowy system mikroprocesorowy, czy to realizowany przy użyciu mikroprocesora, czy mikrokontrolera, zawiera:

- centralną jednostkę procesora (CPU),
- pamięć programu (ROM, PROM, EPROM lub EEPROM),
- pamięć danych (RAM),
- porty wejścia/wyjścia (I/O).

CPU (centralna jednostka procesora)

CPU jest „mózgiem” każdego systemu mikroprocesorowego, ponieważ koordynuje ono działanie wszystkich pozostałych układów, a ponadto wykonuje wszystkie operacje (obliczenia) arytmetyczne i logiczne. Ponadto, podczas wykonywania instrukcji CPU adresuje pamięć i układy wejścia/wyjścia, oraz odpowiada na zewnętrzne sygnały sterujące. Wewnątrz CPU znajdują się, odpowiednio ze sobą połączone:

- rejestry,
- jednostka arytmetyczno-logiczna (ALU),
- układy sterujące (np. dekodery instrukcji),
- układ obsługi przerwań.

Rejestry stanowią rodzaj pamięci do chwilowego przechowywania różnego rodzaju informacji. Niektóre z rejestrów, jak np. licznik rozkazów pełnią specjalne funkcje. Inne, jak np. akumulator, mają bardziej ogólne zastosowanie.

Jednostka arytmetyczno-logiczna (ALU), jak sama nazwa wskazuje, służy do wykonywania obliczeń arytmetycznych i operacji logicznych. Operacje te są zwykle wykonywane na liczbach dwójkowych. Niektóre mikrokontrolery i mikroprocesory umożliwiają wykonywanie operacji także na liczbach dziesiętnych. Jednostka ALU zawiera sumator, który umożliwia dodanie zawartości dwóch rejestrów. Programista może tak skonstruować oprogramowanie, by wykorzystując sam tylko sumator, możliwe było wykonywanie innych operacji arytmetycznych takich jak odejmowanie, mnożenie, dzielenie itp. W praktyce potrzeba tworzenia takiego oprogramowania zachodzi niezbyt często, ponieważ wymienione funkcje arytmetyczne, a także wybrane funkcje logiczne oraz operacje przesuwania zawartości wybranych rejestrów są już z reguły wbudowane (realizowane sprzętowo przez układ ALU). Układ ALU zawiera również specjalne bity tzw. wskaźników, które służą do sygnalizowania określonych stanów, powstałych w wyniku wykonywania operacji arytmetycznych i logicznych. Stan tych wskaźników może być testowany programowo, a wykrycie określonego stanu wybranych wskaźników może być wykorzystane do spowodowania skoku w inne miejsce wykonywanego programu.

Układy sterujące, na podstawie sygnału zegarowego (taktującego), generują odpowiednie sygnały sterujące pobieraniem kodu instrukcji z pamięci programu, dekodowaniem kodu operacji oraz wysyłają do innych wewnętrznych i zewnętrznych układów sygnały niezbędne do wykonania przez CPU kolejnych operacji. Układy sterujące są również odpowiedzialne za obsługę specjalnych sygnałów zewnętrznych – np. przerwań. Zgłoszenie

przerwania powoduje, że układ sterujący zawiesza chwilowo wykonywanie programu głównego i wykonuje skok do procedury obsługi przerwania, a po jej wykonaniu wymusza automatyczny powrót do miejsca, w którym przerwane zostało wykonywanie programu głównego. Przerwania nie zawsze mają postać sygnałów zewnętrznych – mogą być one generowane także przez układy wewnętrzne, np. liczniki, łącza szeregowo.

Stosowanie przerwań umożliwia maksymalne wykorzystanie możliwości przetwarzania danych, jakie daje mikrokontroler. Dość często system mikroprocesorowy współpracuje z urządzeniami stosunkowo wolnymi – jak np. drukarki, wyświetlacze, pamięci masowe. W takich sytuacjach zamiast wytracać czas, oczekując na gotowość urządzenia zewnętrznego do wykonania następnej operacji np. przesłania danych, mikrokontroler może wykonywać inne operacje, a do obsługi danego urządzenia przejść w chwili zgłoszenia przez to urządzenie przerwania sygnalizującego gotowość do dalszej wymiany informacji. Przerwania są również przydatne do obsługi nieregularnych zdarzeń i sytuacji awaryjnych sygnalizowanych z zewnątrz, np. przekroczenia dopuszczalnej wartości ciśnienia płynu, sygnalizowanego przez czujnik pomiarowy. Procedura obsługi takiego przerwania powinna być napisana w taki sposób, by jej wykonanie spowodowało odpowiednią reakcję na sygnalizowane zdarzenie – w podanym przykładzie np. wysłanie sygnału powodującego przymknięcie odpowiedniego zaworu.

Mikrokontroler może współpracować z wieloma urządzeniami zgłaszającymi przerwania. W takim przypadku każdemu z przerwań przypisany jest inny priorytet. Dzięki temu, przy jednoczesnym zgłoszeniu kilku przerwań, w pierwszej kolejności zostanie wykonana procedura obsługi przerwania o najwyższym priorytecie, czyli uznanego za najważniejsze.

Pamięć

Do działania mikrokontrolera niezbędny jest odpowiedni program, przechowywany w nieulotnej pamięci programu. Musi się ona zatem znaleźć w projektowanym systemie i musi być gotowa do odczytu już w chwili załączania zasilania w układzie. Z tego względu pamięć programu wykonywana jest najczęściej jako pamięć typu:

- ROM – programowana maską w fazie produkcji mikrokontrolera,
- PROM – programowana jednorazowo przez użytkownika,
- EPROM, EEPROM – przystosowane do wielokrotnego programowania przez użytkownika.

Do przechowywania wyników wykonywanych operacji i innych danych wykorzystywanych przez CPU konieczne jest zastosowanie pamięci RAM, której zawartość może być zarówno odczytywana, jak i modyfikowana przez CPU.

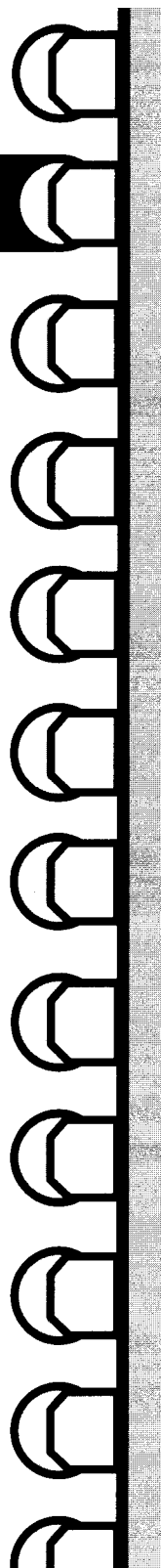
Magistrale i porty wejścia/wyjścia

Większość układów w systemie mikroprocesorowym połączonych jest za pomocą magistral. Są to zespoły połączeń służące do przesyłania sygnałów adresowych, danych itp. wysyłanych i odczytywanych przez CPU.

Porty wejścia/wyjścia są dla mikrokontrolera „mostami” łączącymi mikrokontroler ze światem zewnętrznym. Porty wejściowe umożliwiają odczytywanie informacji z takich źródeł jak na przykład przełączniki i czujniki sygnalizujące zdarzenia zewnętrzne. Porty wyjściowe wykorzystywane są do przesyłania informacji do urządzeń zewnętrznych, takich jak diody LED, przekładniki, silniki, czy nawet inne mikrokontrolery.

**Architektura
mikrokontrolerów
rodziny '51**

2



Możliwości, jakie daje architektura mikroprocesora lub mikrokontrolera nierzadko decydują o wyborze układu, jaki będzie zastosowany w projektowanym systemie. Bardzo duże znaczenie ma tu zwłaszcza organizacja i rozmiar pamięci programu i danych, liczba i możliwości wykorzystania wewnętrznych rejestrów oraz możliwość rozbudowy układu – w szczególności zaś sposób komunikacji z zewnętrznymi układami peryferyjnymi. Na szybkość działania projektowanego systemu bezpośredni wpływ ma oczywiście częstotliwość taktowania mikrokontrolera, określająca czas trwania cyklu zegarowego. Jednakże liczba cykli maszynowych potrzebnych do wykonania konkretnej instrukcji oraz czas trwania cyklu maszynowego (liczony w cyklach zegarowych) zależy już od wewnętrznej architektury mikrokontrolera. W wielu przypadkach o wyborze typu mikrokontrolera decydują możliwości wewnętrznych układów peryferyjnych.

2.1. Organizacja pamięci

Budowa zdecydowanej większości mikrokontrolerów rodziny '51 i ich lista rozkazów pozwalają na korzystanie z:

- pamięci programu o pojemności do 64 KB,
- zewnętrznej pamięci danych o pojemności do 64 KB,
- wewnętrznej pamięci danych o pojemności do 256 bajtów,
- 128-bajtowego obszaru rejestrów specjalnego przeznaczenia SFR (*Special Function Registers*).



W mikrokontrolerach rodziny '51 nie ma podziału na operacje wejścia/wyjścia i operacje na zewnętrznej pamięci danych. Wszystkie zewnętrzne układy peryferyjne oraz zewnętrzna pamięć danych są umieszczane we wspólnym obszarze adresowym nazywanym umownie obszarem zewnętrznej pamięci danych.

2.1.1. Pamięć programu

Prawie wszystkie mikrokontrolery rodziny '51 mogą korzystać z wewnętrznej lub zewnętrznej pamięci programu (tylko nieliczne, jak np. C1051, mają do dyspozycji wyłącznie wewnętrzną pamięć programu). To, z której z tych pamięci pobierane są rozkazy zależy od stanu wyprowadzenia \overline{EA} . W przypadku korzystania z wewnętrznej pamięci programu (programowanej maską w trakcie produkcji – wersje ROM mikrokontrolerów lub przez użytkownika – wersje EPROM/EEPROM mikrokontrolerów) wyprowadzenie \overline{EA} musi być ustawione w stan wysoki. Jeśli pojemność wewnętrznej pamięci programu jest niewystarczająca, część programu może być umieszczona w dodatkowej pamięci zewnętrznej. Wówczas, dopóki wartość licznika rozkazów nie przekracza rozmiaru wewnętrznej pamięci programu, rozkazy pobierane są z pamięci wewnętrznej. Przekroczenie przez licznik rozkazów rozmiaru wewnętrznej pamięci programu powoduje pobieranie rozkazów z zewnętrznej pamięci programu. Jeśli mikrokontroler (z wewnętrzną pamięcią programu lub bez niej) ma korzystać wyłącznie z zewnętrznej pamięci programu, wyprowadzenie \overline{EA} musi być ustawione w stan niski.



W zdecydowanej większości mikrokontrolerów wyprowadzenie \overline{EA} nie ma wewnętrznego rezystora podciągającego i nie może pozostawać niepodłączone („wisieć w powietrzu”).

Zerowanie (*reset*) mikrokontrolera powoduje ustawienie licznika rozkazów w stan 0000h, tak więc początek programu musi być umieszczony pod adresem 0000. Zwykle jest tam umieszczana instrukcja skoku do dalszego obszaru pamięci programu, ponieważ począwszy od adresu 03h pierwsze kilkadziesiąt do stukilkudziesięciu bajtów (w zależności od typu mikrokontrolera) wykorzystywanych jest przez procedury obsługi przerwań.

2.1.2. Wewnętrzna pamięć danych

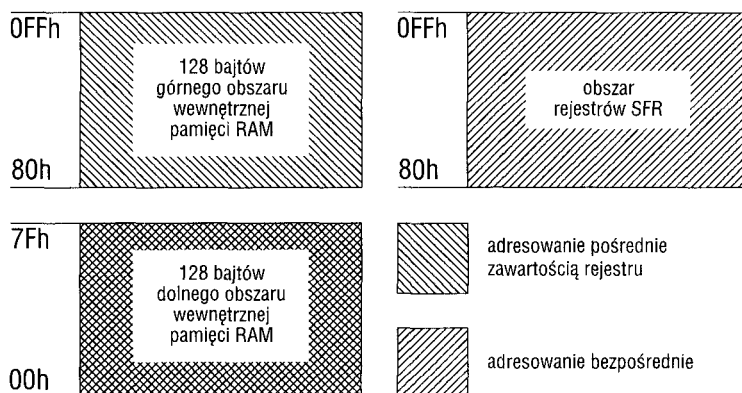
Wewnętrzną pamięć danych tworzą trzy rozłączne bloki (rys. 2.1):

- 128 bajtów tzw. dolnego obszaru pamięci RAM,
- 128 bajtów górnego obszaru pamięci RAM,
- 128-bajtowa przestrzeń adresowa rejestrów specjalnego przeznaczenia, określanych też mianem rejestrów sterujących lub rejestrów SFR.

Nie wszystkie wymienione zasoby muszą być wykorzystane. Wiele mikrokontrolerów nie ma np. pamięci RAM górnego obszaru, niektóre mają mniej niż 128 bajtów pamięci RAM dolnego obszaru, nigdy też nie jest wykorzystany w pełni obszar rejestrów SFR. Odwołania do wewnętrznej pamięci danych realizowane są zawsze na podstawie adresu jednobajtowego, a z powyższego opisu wynika, że wykorzystywana pamięć może zajmować więcej niż 256 bajtów. Z tego powodu dostęp do pokrywających się adresowo górnego obszaru pamięci RAM i rejestrów SFR uzyskuje się stosując różne tryby adresowania (rys. 2.1).

W dolnym obszarze wewnętrznej pamięci RAM (rys. 2.2) można wyróżnić:

- 4 banki rejestrów ogólnego zastosowania (adresy 00h...1Fh),
- 16 bajtów (adresy 20h...2Fh) zawierających 128 bezpośrednio adresowalnych bitów,
- pamięć ogólnego zastosowania (adresy 30h...7Fh).



Rys. 2.1. Organizacja wewnętrznej pamięci RAM mikrokontrolerów rodziny '51

127	Pamięć ogólnego zastosowania								7Fh
48									30h
47	7F	7E	7D	7C	7B	7A	79	78	2Fh
46	77	76	75	74	73	72	71	70	2Eh
45	6F	6E	6D	6C	6B	6A	69	68	2Dh
44	67	66	65	64	63	62	61	60	2Ch
43	5F	5E	5D	5C	5B	5A	59	58	2Bh
42	57	56	55	54	53	52	51	50	2Ah
41	4F	4E	4D	4C	4B	4A	49	48	29h
40	47	46	45	44	43	42	41	40	28h
39	3F	3E	3D	3C	3B	3A	39	38	27h
38	37	36	35	34	33	32	31	30	26h
37	2F	2E	2D	2C	2B	2A	29	28	25h
36	27	26	25	24	23	22	21	20	24h
35	1F	1E	1D	1C	1B	1A	19	18	23h
34	17	16	15	14	13	12	11	10	22h
33	0F	0E	0D	0C	0B	0A	09	08	21h
32	07	06	05	04	03	02	01	00	20h
31	RB3								1Fh
24									18h
23	RB2								17h
16									10h
15	RB1								0Fh
8									8h
7	R7								7h
	R6								6h
	R5								5h
	R4								4h
	R3								3h
	R2								2h
	R1								1h
0	R0								0h

Rys. 2.2. Struktura dolnego obszaru wewnętrznej pamięci RAM

2.1.3. Rejestry ogólnego zastosowania

W mikrokontrolerach rodziny '51 rejestry ogólnego zastosowania GPR (*General Purpose Registers*), określane też mianem rejestrów roboczych, zostały sformowane w cztery banki po osiem rejestrów. Znaczna część instrukcji mikrokontrolerów rodziny '51 może być realizowana w postaci odwołania nie do bezwzględnego adresu pamięci, lecz do rejestru ogólnego zastosowania i odnosi się wówczas do rejestru

o podanym numerze, znajdującego się w aktywnym banku rejestrów. Rozwiązanie to umożliwia przyspieszenie obsługi procedur przerwań itp., ponieważ czasochłonne przysyłanie zawartości rejestrów na stos lub z powrotem można wówczas zastąpić prostą i szybką zmianą aktywnego banku rejestrów. Numer aktywnego banku rejestrów jest określony stanem bitów PSW.3 i PSW.4 znajdujących się w rejestrze wskaźników stanu (PSW). W danej chwili tylko jeden bank rejestrów jest aktywny. Po wyzerowaniu mikrokontrolera bankiem aktywnym jest bank zerowy (RB0).

2.1.4. Stos

Stos jest rodzajem pamięci danych, której charakter bardzo dobrze oddaje jej nazwa. Pobieranie danych ze stosu musi być bowiem wykonywane w kolejności odwrotnej niż ich umieszczanie na stosie – element przesłany na stos jako ostatni musi być zeń zdjęty jako pierwszy. Tak więc w danej chwili możliwy jest dostęp do jednego tylko – ostatniego elementu stosu. Rozmiar stosu ulega ciągłym zmianom, w miarę umieszczania na nim lub pobierania zeń kolejnych bajtów danych.

Podstawowym zastosowaniem stosu jest zapamiętywanie adresów powrotu podczas wywoływania procedur. Bardzo często stos wykorzystywany jest też jako rodzaj podręcznej pamięci do chwilowego przechowywania danych. Może być on również wykorzystywany do przekazywania parametrów procedur, do realizacji przełączania procesów (w systemach wielozadaniowych) itp.

Stos w mikrokontrolerach rodziny '51 jest umieszczony w wewnętrznej pamięci RAM. Położenie końca stosu określa 8-bitowy rejestr SP (wskaźnik stosu). Zwiększanie się stosu jest równoznaczne z zapisywaniem kolejnych bajtów pamięci o adresach wzrastających. Po wyzerowaniu mikrokontrolera wskaźnik stosu przyjmuje wartość 07h, a pierwszy bajt stosu jest zapisywany pod adresem 08h, czyli w zerowym (R0) rejestrze pierwszego banku rejestrów (RB1). Stos może być przeniesiony w dowolne miejsce wewnętrznej pamięci RAM przez wpisanie odpowiedniej wartości do rejestru SP. Stosu nie można umieścić w zewnętrznej pamięci RAM.

2.1.5. Zewnętrzna pamięć danych

Prawie wszystkie mikrokontrolery rodziny '51 mogą korzystać z zewnętrznej pamięci danych o pojemności do 64 KB. Możliwości tej pozbawione są jedynie mikrokontrolery o zredukowanej liczbie wyprowadzeń (np. C2051) oraz niektóre mikrokontrolery specjalizowane (np. C055, C852). Zewnętrzna pamięć danych (oraz zewnętrzne układy peryferyjne) w mikrokontrolerach rodziny '51 może być adresowana wyłącznie pośrednio – za pomocą 16-bitowego wskaźnika danych DPTR (adresowanie w zakresie 64 KB) lub 8-bitowych rejestrów R0 i R1 z aktywnego banku rejestrów (adresowanie w zakresie 256 bajtów). Adresowanie pośrednie polega na tym, że adres nie jest podawany jawnie (w postaci stałej lub etykiety występującej w instrukcji), lecz jest określany przez mikrokontroler na bieżąco, na podstawie zawartości odpowiedniego rejestru (wymienionego w instrukcji). Mikrokontrolery rodziny '51 mają architekturę 8-bitową – w związku z tym konieczne było opracowanie specjalnych metod umożliwiających zmianę zawartości 16-bitowego wskaźnika danych DPTR. Zawartość tego wskaźnika może być modyfikowana przy użyciu specjalnych, 16-bitowych rozkazów

MOV DPTR, #stała oraz INC DPTR, bądź przez zmianę zawartości starszego lub młodszego bajtu wskaźnika. Obie połówki wskaźnika DPTR widziane są przez CPU jako 8-bitowe rejestry: DPL (młodszy bajt) i DPH (starszy bajt) umieszczone w przestrzeni rejestrów SFR.

Istnienie tylko jednego 16-bitowego wskaźnika DPTR jest dosyć często czynnikiem istotnie ograniczającym szybkość wymiany informacji z zewnętrzną pamięcią danych. Mając na celu jej usprawnienie, w niektórych mikrokontrolerach wprowadzony został zestaw kilku wskaźników DPTR. Założenie o stuprocentowej programowej kompatybilności z mikrokontrolerem 8051 równoznaczne jest jednak z korzystaniem w danej chwili z jednego tylko rejestru DPTR. Z tego względu omawiane mikrokontrolery, oprócz zestawu wskaźników DPTR, wyposażone są zwykle w dodatkowy rejestr sterujący (oznaczany jako DPSEL lub DPS, choć w niektórych układach rejestr ten miewa też inne nazwy, np. AUXR1) – zawartość tego rejestru określa, który ze wskaźników DPTR jest w danej chwili wskaźnikiem aktywnym. W niektórych mikrokontrolerach bit sterujący wyborem aktywnego wskaźnika DPTR może być umieszczony „gościinnie” w rejestrze przeznaczonym w zasadzie do innych celów (np. w układzie S8252 rolę tę pełni bit DPS w rejestrze WMCON, wykorzystywanym głównie do sterowania układem licznika czuwającego). Wybór aktywnego wskaźnika DPTR odbywa się zatem przez przypisanie odpowiedniej wartości wspomnianym bitom. Po zmianie zawartości rejestru DPSEL (czy też odpowiedniego innego rejestru zawierającego omawiane bity sterujące) wszystkie instrukcje odwołujące się do wskaźnika danych będą wykorzystywały wskaźnik DPTR (ewentualnie tylko jego młodszy lub starszy bajt) wyznaczony przez nową zawartość rejestru (bitów sterujących). W przypadku mikrokontrolerów wyposażonych w zestaw dwóch wskaźników DPTR młodsze i starsze bajty obu wskaźników (DPL, DPL1, DPH, DPH1) są zwykle bezpośrednio dostępne w obszarze rejestrów SFR – dzięki temu operacje działające na połówkach obu wskaźników DPTR można wykonywać bez konieczności odpowiednich zmian zawartości rejestru DPSEL.

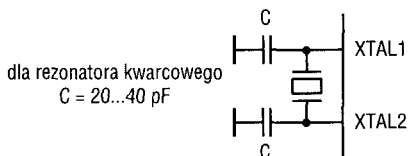
2.1.6. Rejestry specjalnego przeznaczenia (SFR)

Obszar rejestrów specjalnego przeznaczenia w mikrokontrolerach rodziny '51 wykorzystywany jest dwojako. Z jednej strony umieszczone są w nim wszystkie (z wyjątkiem licznika rozkazów i czterech banków rejestrów ogólnego zastosowania) rejestry sterujące pracą lub wykorzystywane bezpośrednio przez CPU – akumulator ACC, akumulator pomocniczy B, rejestr wskaźników stanu PSW, wskaźnik stosu SP i pozostałe rejestry wskaźnikowe. Z drugiej zaś strony rejestry SFR stanowią rodzaj interfejsu pomiędzy CPU a układami peryferyjnymi umieszczonymi wewnątrz mikrokontrolera. Wszystkie operacje sterowania wewnętrznymi układami peryferyjnymi oraz przesyłania danych między nimi a CPU odbywają się właśnie za pośrednictwem rejestrów SFR. Rejestry SFR są umieszczone w obszarze wewnętrznej pamięci danych o adresach 80h...0FFh, a dostęp do każdego z tych rejestrów jest realizowany wyłącznie w trybie adresowania bezpośredniego (patrz rozdz. 2.1.2). Wszystkie rejestry umieszczone pod adresami o wartościach podzielnych przez osiem dostępne są także bitowo, przy czym adres n -tego (licząc od zera) bitu rejestru spod adresu m jest równy $m+n$ (np. trzeci bit akumulatora A ma adres 0E0h + 03h = 0E3h). Niektóre spośród rejestrów SFR, które nie są wykorzystywane zgodnie z ich pierwotnym

przeznaczeniem, tj. do sterowania konkretnymi funkcjami układów peryferyjnych itp., mogą być używane jako pamięć ogólnego zastosowania o dostępie bezpośrednim.

2.2. Taktowanie CPU

2.2.1. Generator sygnału zegarowego



Rys. 2.3. Schemat obwodów zewnętrznych generatora taktującego z rezonatorem kwarcowym lub ceramicznym

Wszystkie mikrokontrolery rodziny '51 mają wbudowany generator sygnału zegarowego, który może być wykorzystany jako źródło sygnału taktującego mikrokontroler. W tym celu do wyprowadzeń XTAL1 i XTAL2 mikrokontrolera wystarczy podłączyć rezonator kwarcowy lub ceramiczny oraz

dwa kondensatory (rys. 2.3). Wartość pojemności kondensatorów nie jest krytyczna – zwykle stosuje się wartości około 20...40 pF. W przypadku rezonatora ceramicznego pojemności kondensatorów powinny być nieco większe – zwykle dołącza się kondensatory 47 pF.

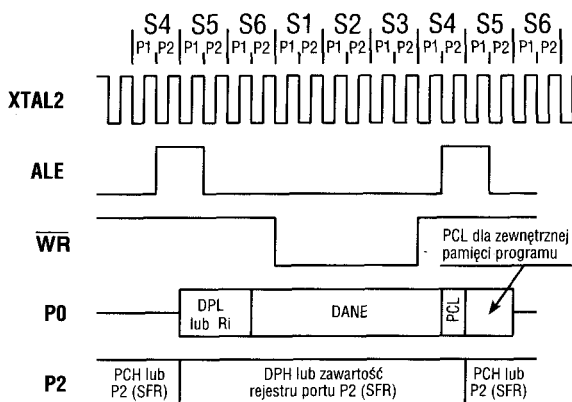
Omawiany układ generatora sygnału zegarowego powoduje wzbudzenie rezonatora na jego podstawowej częstotliwości rezonansu. Stanowi to istotny problem jeśli projektowany system ma być taktowany z dużą częstotliwością (np. 30 MHz), ponieważ dla takich częstotliwości rezonatory produkowane są niemal wyłącznie jako owerotonowe. Rezonator taki można wprawdzie zmusić do pracy na częstotliwości harmonicznej, jednakże wymaga to z reguły stosowania dość kłopotliwych zabiegów, jak np. stosowanie dodatkowych cewek. W takich przypadkach znacznie wygodniejszym rozwiązaniem jest użycie scalonego kwarcowego generatora częstotliwości. Generatory takie przystosowane są do zasilania napięciem 5 V, produkowane są w czterokońcówkowych obudowach, a ich sygnał wyjściowy jest zgodny ze standardem TTL/CMOS. Wykorzystanie takich generatorów jest możliwe, ponieważ mikrokontrolery rodziny '51 mogą być także taktowane sygnałem zewnętrznym. W przypadku taktowania mikrokontrolera sygnałem zewnętrznym pojawiają się jednak między producentami pewne rozbieżności co do wykorzystania wyprowadzeń XTAL1 i XTAL2 oraz parametrów sygnału taktującego. Standardowo w mikrokontrolerach NMOS taktowanych sygnałem zewnętrznym sygnał taktujący należy podłączyć na wyprowadzenie XTAL2, a wyprowadzenie XTAL1 połączyć z masą. W mikrokontrolerach CMOS wyprowadzenie, na które należy podać sygnał taktujący zmienia się w zależności od producenta i typu układu. Zdarza się nawet i tak, że ta sama firma produkuje dwie niemal identyczne wersje tego samego mikrokontrolera, a różnica dotyczy między innymi tego, które z wyprowadzeń XTAL1, XTAL2 ma być wykorzystywane do taktowania mikrokontrolera sygnałem zewnętrznym (ma to np. miejsce w przypadku mikrokontrolerów 80C51 produkowanych przez firmę OKI). Zawsze jednak drugie z wyprowadzeń XTAL1, XTAL2 w mikrokontrolerach rodziny '51 wersji CMOS należy pozostawić niepodłączone. Jeśli w projektowanym systemie mają być wyko-

rzystywane mikrokontrolery różnych producentów lub w momencie projektowania nie wiadomo, które z wyprowadzeń XTAL1, XTAL2 należy przyjąć za aktywne, to najprostszym i skutecznym rozwiązaniem jest dołączenie sygnału taktującego za pośrednictwem zwory, umożliwiającej wybór aktywnego wyprowadzenia XTAL1, XTAL2. Przy taktowaniu mikrokontrolera sygnałem zewnętrznym rozbieżności dotyczą też parametrów sygnału taktującego. Najczęściej można przyjąć, że przy taktowaniu sygnałem zewnętrznym napięcie w stanie niskim na wyprowadzeniu XTAL nie powinno przekraczać 0,75 V (zarówno w mikrokontrolerach NMOS jak i CMOS), a w stanie wysokim nie powinno być niższe niż 2,5 V w przypadku mikrokontrolerów odmiany NMOS oraz 3,5 V w przypadku mikrokontrolerów wersji CMOS. Z tego względu do sterowania wyprowadzenia XTAL najwygodniej jest zastosować bramkę o poziomach logicznych CMOS. Niejednoznaczności dotyczą też wypełnienia sygnału taktującego. Zwykle może być ono różne od 50%, ponieważ sygnał wejściowy jest podawany na przerzutnik dzielący przez dwa, z którego otrzymywany jest już sygnał o 50-procentowym wypełnieniu. Niektórzy producenci mikrokontrolerów wymagają jednak, aby zewnętrzny sygnał taktujący miał wypełnienie równe 50%.

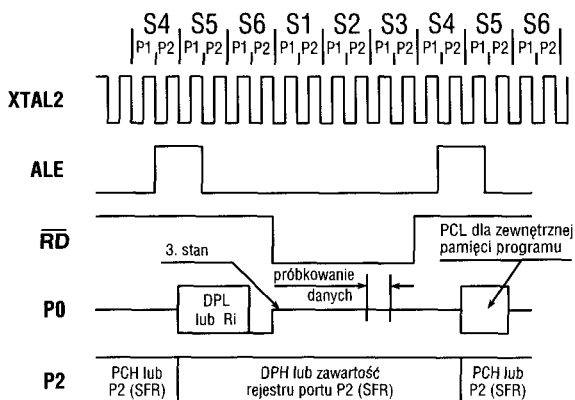
2.2.2. Taktowanie mikrokontrolera

Cykl maszynowy standardowych mikrokontrolerów rodziny '51 składa się z sześciu tzw. stanów, oznaczanych symbolami S1 do S6. Każdy ze stanów trwa dwa okresy sygnału taktującego (cykle zegarowe) nazywane fazami i oznaczane jako P1 i P2. Tak więc na pojedynczy cykl maszynowy składa się 12 taktów (okresów) sygnału zegarowego, co oznacza, że przy sygnale taktującym o częstotliwości 12 MHz każdy cykl maszynowy będzie trwał 1 μ s. W ciągu ostatnich lat pojawiły się wprowadzić w rodzinie '51 liczne mikrokontrolery, w których cykl maszynowy składa się z zaledwie 6, 4, czy nawet z 1 cyklu zegarowego, nie dotyczy to jednak standardowych mikrokontrolerów C51, C52, a także wielu nowszych – jak np. S8252, czy C2051.

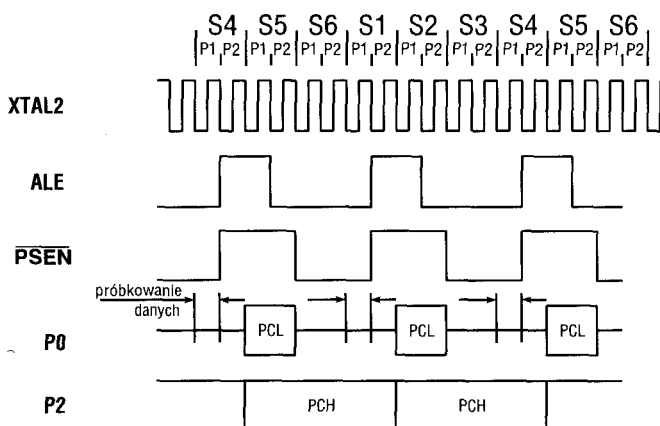
Czas wykonywania instrukcji standardowych mikrokontrolerów rodziny '51 wynosi – w zależności od wykonywanej instrukcji – jeden, dwa lub cztery cykle maszynowe. Tylko dwie spośród wszystkich instrukcji mikrokontrolerów rodziny '51 są wykonywane w ciągu 4 cykli maszynowych – są to instrukcje mnożenia i dzielenia.



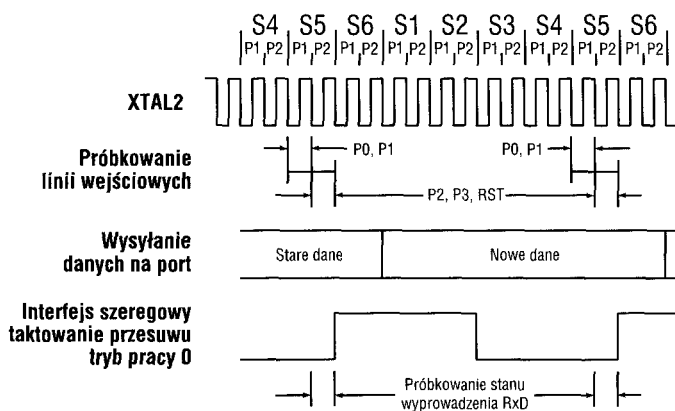
Rys. 2.4. Taktowanie mikrokontrolera podczas zapisu do zewnętrznej pamięci RAM



Rys. 2.5. Taktowanie mikrokontrolera podczas odczytu zewnętrznej pamięci RAM



Rys. 2.6. Taktowanie mikrokontrolera podczas odczytu zewnętrznej pamięci programu



Rys. 2.7. Taktowanie mikrokontrolera podczas operacji na portach

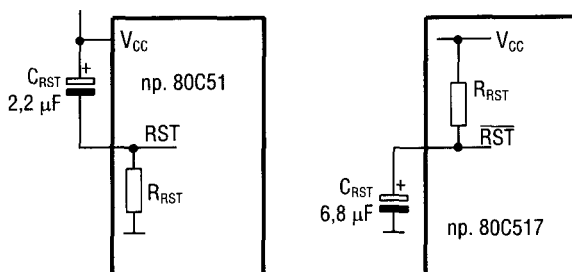
Większość spośród instrukcji mikrokontrolerów rodziny '51 jest wykonywana w ciągu jednego cyklu maszynowego. Podział cyklu maszynowego na stany, w ramach których wyróżniane są jeszcze fazy wynika z konieczności generowania odpowiednich sygnałów, zarówno wewnętrznych jak i zewnętrznych, które sterują poszczególnymi etapami wykonywania instrukcji. W ramach każdej instrukcji musi bowiem wystąpić etap pobrania kodu instrukcji z pamięci programu, zdekodowania pobranego kodu i wykonania operacji, w ramach której można wyróżnić pobranie dodatkowych argumentów, zapis wyniku do odpowiedniego rejestru itp. Taktowanie i generowanie zewnętrznych sygnałów sterujących na przykładzie operacji pobrania instrukcji z zewnętrznej pamięci programu, zapisu i odczytu zewnętrznej pamięci danych oraz odczytu/zapisu portów mikrokontrolera pokazano na rys. 2.4...2.7 (przebiegi te mogą zmieniać się nieznacznie, w zależności od producenta danego mikrokontrolera).

2.3. Zerowanie mikrokontrolera (RESET)

Praktycznie wszystkie mikrokontrolery rodziny '51 mają specjalne wyprowadzenie służące do zerowania mikrokontrolera. Zerowanie polega na wykonaniu przez mikrokontroler szeregu operacji, wskutek których znaczna część wewnętrznych rejestrów SFR przyjmuje określone przez producenta wartości. Dzięki temu początkowe warunki pracy mikrokontrolera po jego wyzerowaniu są zawsze jednakowe. W szczególności w wyniku zerowania mikrokontrolera licznik programu przyjmuje wartość 0000h, wskaźnik stosu wartość 07h, a rejestry wyjściowe portów wartość 0FFh. Oznacza to, że kod pierwszej instrukcji wykonanej po zakończeniu operacji zerowania zostanie pobrany spod adresu 0000h pamięci programu, pierwszy bajt umieszczony na stosie znajdzie się pod adresem 08h wewnętrznej pamięci RAM mikrokontrolera, a wszystkie linie portów zostaną ustawione w stan wysoki (z tym, że np. linie portu P0 przyjmą stan wysokiej impedancji ze względu na brak wewnętrznych rezystorów podciągających). W wyniku zerowania rejestry sterujące pracą wewnętrznych układów peryferyjnych mikrokontrolera przyjmują z reguły takie wartości, by dany układ pozostał nieaktywny. Zerowanie mikrokontrolera nie powoduje zmian zawartości wewnętrznej pamięci RAM.

UWAGA

W przypadku niemal wszystkich mikrokontrolerów rodziny '51 zerowanie powoduje ustawienie linii portów w stan wysoki (można to uznać za regułę w przypadku wszystkich najbardziej popularnych układów). Istnieją jednak mikrokontrolery, w których standardowa wartość wpisywana do rejestrów portów podczas zerowania mikrokontrolera jest różna od 0FFh. Ponadto należy pamiętać, że np. od chwili załączenia zasilania, do zakończenia wykonywania wewnętrznej operacji zerowania, stan logiczny wyprowadzeń mikrokontrolera może być nieokreślony. Tylko w przypadku tych mikrokontrolerów, które wyposażone są w układ asynchronicznego ustawiania linii portów, linie portów przyjmują stan standardowy niemal natychmiast (tj. w czasie kilku – kilkadziesiąt mikrosekund) po załączeniu zasilania.



Rys. 2.8. Sposoby dołączenia zewnętrznego kondensatora (w obwodzie automatycznego zerowania po załączeniu zasilania) w zależności od polaryzacji sygnału zerującego

Operacja zerowania trwa dwa cykle maszynowe i jest wykonywana w odpowiedzi na aktywny sygnał zerowania na wyprowadzeniu RESET (oznaczanym też czasami symbolem RST). Stan wyprowadzenia RESET jest testowany tylko jeden raz (w stanie S5P2) w ciągu każdego cyklu maszynowego. Dla poprawnego przeprowadzenia operacji zerowania aktywny poziom sygnału RESET musi trwać co najmniej dwa pełne cykle maszynowe. Stwierdzenie to jest jednak słuszne tylko przy założeniu, że w chwili pojawienia się sygnału zerowania generator sygnału zegarowego mikrokontrolera pracuje normalnie. Jeśli sygnał zerowania pojawia się w chwili gdy generator nie działa, co ma miejsce np. po włączeniu napięcia zasilającego lub przy wychodzeniu ze stanu zamrożenia (jeśli mikrokontroler nie jest taktowany sygnałem zewnętrznym), to sygnał zerujący musi być w stanie aktywnym przez co najmniej dwa cykle maszynowe licząc od chwili startu generatora. Biorąc pod uwagę, że przy stosowaniu rezonatora kwarcowego typowy czas rozruchu generatora jest rzędu milisekundy, bezpieczna wartość czasu trwania impulsu zerującego wynosi co najmniej 10...20 ms. Utrzymywanie wyprowadzenia zerującego w stanie aktywnym dłużej niż dwa cykle maszynowe nie pociąga za sobą żadnych skutków, poza ciągłym powtarzaniem cykli zerowania. Powrót wyprowadzenia RESET do stanu nieaktywnego powoduje rozpoczęcie normalnej pracy mikrokontrolera. Pierwsze opadające zbocze sygnału ALE pojawia się w stanie S5P2, nie wcześniej niż jeden cykl i nie później niż dwa cykle maszynowe od momentu powrotu sygnału zerującego do stanu nieaktywnego.

Po włączeniu zasilania zawartość zarówno wewnętrznej pamięci RAM, jak i rejestrów mikrokontrolera jest nieokreślona. Z tego względu zerowanie mikrokontrolera po włączeniu napięcia zasilania jest warunkiem koniecznym prawidłowej pracy mikrokontrolera. W przypadku mikrokontrolerów rodziny '51 realizacja automatycznego zerowania mikrokontrolera w wyniku załączenia napięcia zasilania jest niezwykle prosta i polega na podłączeniu kondensatora na wyprowadzenie RESET (rys. 2.8). Wartość stosowanego kondensatora wynosi zwykle kilka mikrofaradów. Wybór okładki kondensatora podłączanej na wyprowadzenie RESET (z racji znacznej pojemności jest to zwykle kondensator elektrolityczny) zależy od aktywnego poziomu sygnału zerującego, który nie jest niestety jednakowy dla wszystkich mikrokontrolerów rodziny '51 (patrz tab. 2.1). Drugą okładkę kondensatora zerującego należy podłączyć do masy w przypadku mikrokontrolerów o aktywnym niskim poziomie sygnału zerowania, a do napięcia zasilania - w przypadku zerowania poziomem wysokim (C51, C52, S8252, C2051 itp.). Zewnętrzny rezystor z reguły

nie jest potrzebny, gdyż wszystkie mikrokontrolery rodziny '51 wykonane w technologii CMOS mają wewnętrzny rezystor o wartości z zakresu 20...300 k Ω (tab. 2.1). Stosowanie dodatkowego, zewnętrznego rezystora mogłoby mieć sens, gdyby chodziło o względnie precyzyjne ustalenie czasu trwania impulsu zerowania (gdyż wartość wewnętrznego rezystora ma znaczny rozrzut technologiczny). Biorąc jednak pod uwagę duże tolerancje wartości kondensatorów elektrolitycznych, jakie typowo stosuje się w takich układach, precyzja uzyskana przy użyciu owego zewnętrznego rezystora nadal będzie niewielka.

W niektórych mikrokontrolerach (np. firmy Dallas Semiconductor) realizacja automatycznego zerowania po załączeniu zasilania jest jeszcze prostsza, ponieważ nie wymaga stosowania żadnych elementów zewnętrznych. Mikrokontrolery te mają wbudowany układ monitorowania napięcia zasilania oraz specjalny licznik opóźniający. Dopóki wartość napięcia zasilania jest niższa od pewnej wartości progowej V_{RST} , układ monitorowania utrzymuje mikrokontroler w stanie nieaktywnym. Po przekroczeniu przez napięcie zasilania wartości V_{RST} , uruchamiany jest układ generatora sygnału zegarowego, a impulsy z tego generatora zliczane są przez 16-bitowy licznik opóźniający. Dopiero przepelnienie tego licznika (czyli zliczenie 65536 impulsów z generatora sygnału zegarowego) powoduje wyjście mikrokontrolera ze stanu zerowania. Jeśli w trakcie zliczania napięcie zasilania

Tab. 2.1. Wybrane parametry najczęściej wykorzystywanych mikrokontrolerów rodziny '51

Typ	Aktywny poziom sygnału zerującego	R_{RST} [k Ω]	Zalecana wartość C_{RST} [μ F]	Wyjście zerowania	Asynchroniczne ustawianie linii portów	Maksymalna częstotliwość f_{osc} [MHz]	Minimalny czas trwania cyklu maszynowego [ns]	Wyjście zegara systemowego	Dodatkowe wskaźniki DPTR
C51	wysoki	20...300	1 (Intel)	-	+ (OKI, nowsze układy Philipsa)	40	250	T2 (P1.0) (nowsze układy Philipsa)	+ (nowsze układy Philipsa)
LV51	wysoki	50...300	1	-	-	12	1000	-	-
S51	wysoki	40...300	1	+	-	33	364	T2 (P1.0)	+
C52	wysoki	40...300	1 (Intel)	-	+ (nowsze układy Philipsa)	33	364	T2 (P1.0) (większość nowszych układów)	+ (nowsze układy Philipsa)
LV52	wysoki	50...300	1	-	-	12	1000	T2 (P1.0)	-
S52	wysoki	40...300	1	+	-	33	364	T2 (P1.0)	+
S53	wysoki	40...300	1	+	-	33	364	T2 (P1.0)	+
C1051	wysoki	50...300	2,2	-	+	24	500	-	-
C2051	wysoki	50...300	2,2	-	+	24	500	-	-
C4051	wysoki	50...300	2,2	-	+	24	500	-	-
S8252	wysoki	50...300	2,2	-	+	24	500	T2 (P1.0)	+
LS8252	wysoki	50...300	2,2	-	+	12	1000	T2 (P1.0)	+

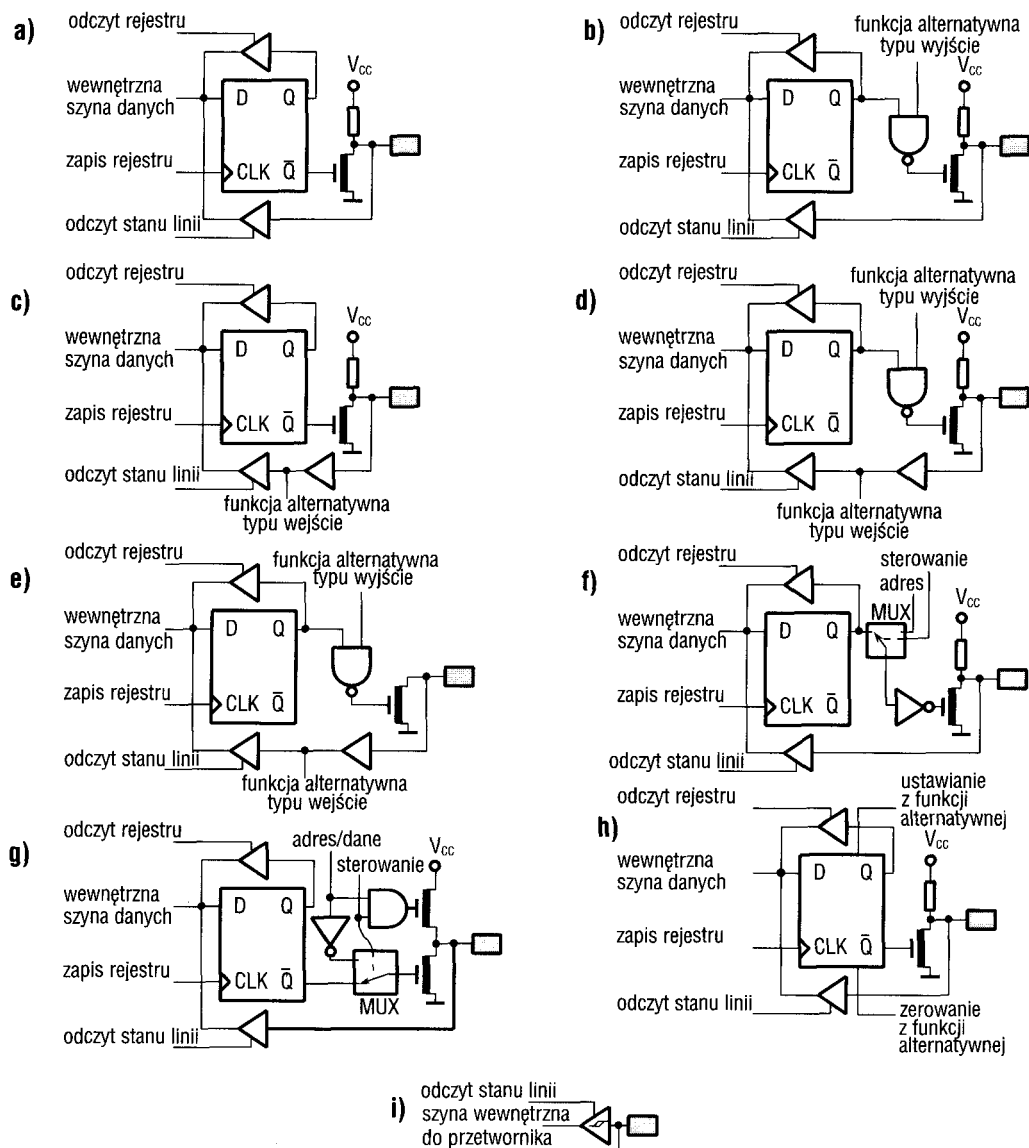
spadnie poniżej wartości V_{RST} , licznik opóźniający jest zerowany i cała operacja jest powtarzana od początku. Zastosowanie takiego rozwiązania gwarantuje skuteczne wyzerowanie mikrokontrolera i uzyskanie stabilnego sygnału z generatora. Wyprowadzenie zerowania mikrokontrolerów firmy Dallas Semiconductor posiada wewnętrzny rezystor połączony z masą. Tak więc jeśli w projektowanym układzie nie przewiduje się zerowania mikrokontrolera sygnałem zewnętrznym (zerowanie mikrokontrolera ma być wykonywane tylko po załączeniu zasilania), wejście sygnału zerowania można pozostawić niepodłączone.

Zerowanie wskutek załączenia napięcia zasilającego nie jest jedynym przypadkiem zerowania mikrokontrolera. W niektórych mikrokontrolerach zerowanie sygnałem zewnętrznym jest np. jedyną drogą do wyjścia ze stanu zamrożenia. Powodem wykonania operacji zerowania może być też np. zadziałanie (wewnętrznego lub zewnętrznego) układu czuwającego. Zerowanie jest wówczas reakcją na błędne działanie oprogramowania lub sprzętu, mającą na celu usunięcie lub przynajmniej minimalizację skutków awarii systemu. W przypadkach gdy potencjalnych przyczyn zerowania mikrokontrolera jest kilka (załączenie napięcia zasilania, licznik czuwający itp.), pożądaną cechą układu jest możliwość określenia, która z tych przyczyn spowodowała wykonanie wewnętrznej procedury zerowania. Nie nastręcza to większych problemów, ponieważ wykrycie sytuacji awaryjnej przez wewnętrzne układy czuwające spotykane w mikrokontrolerach rodziny '51 polega zazwyczaj na ustawieniu odpowiedniego wskaźnika (bitu) i wygenerowaniu przerwania lub przejściu do operacji zerowania mikrokontrolera. Przyczynę zerowania można zatem określić na podstawie analizy stanu kilku bitów umieszczonych w rejestrach SFR.

2.4. Budowa portów

Jednym z istotniejszych parametrów mikrokontrolerów jednoukładowych, który często decyduje o wyborze układu, jest liczba i możliwości wykorzystania linii wejść/wyjść mikrokontrolera. W celu uelastyczenia mikrokontrolera, a tym samym poszerzenia możliwości jego zastosowań, przy jednoczesnym zachowaniu małych rozmiarów układu, znaczna część linii wejść/wyjść mikrokontrolerów może zwykle pełnić różne funkcje. W przypadku mikrokontrolerów rodziny '51 większości linii wejść/wyjść przypisane są dwie lub trzy alternatywne funkcje. Z reguły, oprócz funkcji zwykłego wejścia/wyjścia cyfrowego, linia może być wejściem lub wyjściem sygnałów strobujących, impulsowych, analogowych, adresu, danych itp. Rodzaj funkcji pełnionej przez daną linię prawie zawsze może być ustalany programowo. Linie pełniące zbliżone funkcje zebrane są w grupy określane mianem portów. W przypadku mikrokontrolerów rodziny '51, z racji ich 8-bitowej architektury, większość portów składa się z ośmiu linii wejść/wyjść.

Budowa portu w znacznym stopniu zależy od funkcji jakie mogą pełnić linie wejść/wyjść wchodzące w skład portu. Jeśli linie te funkcjonują wyłącznie jako wejścia/wyjścia cyfrowe, to są zbudowane jak to pokazano na rys. 2.9a. Jeśli linia pracuje jako wyjście cyfrowe, zapis jedynki logicznej do przerzutnika powoduje odcięcie tranzystora i pojawienie się na wyprowadzeniu zewnętrznym mikrokontrolera stanu wysokiego, na skutek działania rezystora podciągającego przyłączonego do dodatniego napięcia zasilania mikrokontrolera. Zapis zera do przerzutnika powoduje wejście tranzystora w przewodzenie i wymuszenie na wyprowadzeniu zewnętrznym mikrokontrolera stanu



Rys. 2.9. Budowa linii wejść/wyjść mikrokontrolerów rodziny '51: a) linia zwykłych wejść/wyjść cyfrowych; b) linia z alternatywną funkcją wyjścia; c) linia z alternatywną funkcją wejścia; d) linia z alternatywnymi funkcjami wejścia i wyjścia; e) jak w pkt. d) z wyjściem typu otwarty dren; f) linia portu P2; g) linia portu P0; h) linia mogąca pełnić funkcję alternatywną, sterującą bezpośrednio przerzutnikiem stanu wyjściowego linii; i) linia wejścia analogowego z alternatywną funkcją wejścia cyfrowego

niskiego. Aby uniknąć zapamiętywania stanu wyjść w pamięci RAM, co mogłoby być nieodzowne, gdyby informacja o stanie wyjść miała być wykorzystywana w przyszłości, układ wyjściowy został wyposażony w bufor trójstanowy umożliwiający odczyt zawartości przerzutnika.



Wpisanie jedynki do przerzutnika jest warunkiem koniecznym, aby linia mogła pracować jako wejście cyfrowe. Wpisanie zera może doprowadzić do konfliktu wyjść. Konflikt ten nie musi wprowadzić spowodować uszkodzenia mikrokontrolera ani układu zewnętrznego (ze względu na stosunkowo małą wydajność prądową wyjść mikrokontrolera) jednakże uniemożliwi odczyt prawidłowej wartości stanu linii.

W przypadku, gdy linia ma pracować jako wejście cyfrowe, wpisanie jedynki do przerzutnika powoduje, że jedynym obciążeniem wyprowadzenia mikrokontrolera staje się rezystor podciągający. Rezystancja tego rezystora wynosi kilkadziesiąt (50...100) kiloomów, co dla układów cyfrowych jest obciążeniem minimalnym. Stan wejścia jest odczytywany przy wykorzystaniu drugiego bufora trójstanowego.

W mikrokontrolerach rodziny '51 budowa pojedynczej linii wejścia/wyjścia (np. jak na **rys. 2.9a**) jest powielona tak, że każde osiem takich bloków tworzy port. Port widziany jest od strony CPU jako 8-bitowy rejestr SFR, w którym każdy z bitów odpowiada stanowi jednej linii wejść/wyjść. Sterowanie i testowanie stanu linii sprowadza się zatem do zmiany, bądź odczytu zawartości rejestru portu.

W przypadku, gdy linia oprócz zwykłego cyfrowego wejścia/wyjścia może pełnić funkcję alternatywną typu wyjście (np. linie strobów zapisu \overline{WR} i odczytu pamięci \overline{RD} zewnętrznej), budowa pokazana na **rys. 2.9a** ulega niewielkiej modyfikacji przyjmując postać jak na **rys. 2.9b**. Aby linia mogła pełnić funkcję alternatywną, wyjście Q przerzutnika musi być ustawione w stan wysoki, podobnie jak w przypadku gdy linia ma pracować jako wejście cyfrowe. Zapis zera logicznego do przerzutnika, w czasie gdy linia pełni funkcję alternatywną spowoduje fałszowanie informacji niesionej tą linią.

Wpisanie jedynki logicznej do przerzutnika jest warunkiem poprawnej pracy układu także w przypadku, gdy linia wejście/wyjście ma pełnić funkcję alternatywną typu wejście (**rys. 2.9c**). Tę samą właściwość mają także linie wejść/wyjść zrealizowane w sposób pokazany na **rys. 2.9d** i **2.9e**. Stan przerzutników, a tym samym rejestrów portów nie ma natomiast wpływu na poprawność działania funkcji alternatywnych w przypadku portów P0 i P2 (wykorzystywanych do wybierania adresów i przesyłania danych podczas wymiany informacji między mikrokontrolerem, a urządzeniem zewnętrznym), ponieważ funkcje alternatywne tych portów realizowane są z wykorzystaniem multiplexerów (**rys. 2.9f, g**).

Budowa układu końcowego linii wejść/wyjść, który na **rys. 2.9** został przedstawiony jako tranzystor z rezystorem podciągającym, jest w rzeczywistości nieco bardziej skomplikowana (**rys. 2.10**). Zapis do przerzutnika następuje pod koniec cyklu maszynowego (S6P2), a wpisana wartość pojawia się na wyjściu linii portu na początku następnego cyklu maszynowego (S1P1). Dzieje się tak, ponieważ wyjściowy bufor linii portu sprawdza stan przerzutnika tylko podczas pierwszej fazy każdego cyklu zegarowego, a przez czas trwania drugiej fazy utrzymuje stan, który wykrył podczas fazy poprzedniej. Zmiana zawartości przerzutnika ze stanu niskiego na wysoki powoduje włączenie tranzystora T1, który pozostaje aktywny przez całe dwa cykle zegarowe (S1P1, S1P2). Tranzystor T1 ma dużą wydajność prądową – spełnia rolę rezystora podciągającego

Podobnie sprawa przedstawia się w przypadku linii P1.0 i P1.1 w układach C1051, C2051 i C4051, które są podłączone do wewnętrznych komparatorów analogowych i nie zostały wyposażone w wewnętrzne rezystory podciągające. Tak więc jeśli linie te mają funkcjonować jako zwykłe wejścia lub wyjścia cyfrowe, również może zachodzić potrzeba dołączenia do nich zewnętrznych rezystorów podciągających.

Zastosowane w mikrokontrolerach rodziny '51 standardowe rozwiązanie stopnia końcowego (z rezystorem podciągającym) może się w pierwszej chwili wydać znacznie gorsze od stopnia komplementarnego mającego możliwość wprowadzania w stan wysokiej impedancji. Zaletą przyjętego rozwiązania jest jednak prostota sterowania, gdyż przy założeniu możliwości niezależnego konfigurowania każdej linii jako wejście lub wyjście, zastosowanie stopnia komplementarnego wymagałoby użycia dwukrotnie większej liczby przerzutników (stan każdej linii – wysoki, niski lub wysokiej impedancji – musiałby być kodowany na dwóch bitach), a tym samym skomplikowałoby programowanie pracy linii wejść/wyjść.

2.5. Współpraca mikrokontrolera z układami zewnętrznymi

Liczba różnych mikrokontrolerów produkowanych w ramach rodziny '51 jest tak duża, a ich wewnętrzne układy peryferyjne tak zróżnicowane, że w znacznej liczbie rozwiązań praktycznych mikrokontroler pracuje jako jedyny układ w całym systemie i nie wykorzystuje on żadnych dodatkowych układów peryferyjnych. Niekiedy zachodzi jednak konieczność zastosowania specjalizowanych układów peryferyjnych, dodatkowej, zewnętrznej pamięci programu lub danych itp. Konstrukcja mikrokontrolerów rodziny '51 umożliwia łatwe dołączanie takich układów do projektowanego systemu. Należy jednak pamiętać, że nie wszystkie mikrokontrolery są przystosowane do współpracy z zewnętrzną pamięcią programu, czy danych (dotyczy to np. układów C1051, C2051, C4051). Nie oznacza to wprawdzie, że korzystanie z zewnętrznych układów peryferyjnych jest w takim przypadku zupełnie niemożliwe, jednak wymaga ono programowej implementacji wszelkich operacji dostępu do tych układów.

2.5.1. Komunikacja z zewnętrznymi układami peryferyjnymi

Mikrokontrolery rodziny '51 zostały zaprojektowane w taki sposób, że w przypadku pobierania rozkazów z wewnętrznej pamięci programu, prawie wszystkie wyprowadzenia mikrokontrolera mogą być wykorzystane jako linie wejścia/wyjścia. Jednakże w przypadku, gdy zachodzi potrzeba dołączenia do mikrokontrolera zewnętrznej pamięci programu, pamięci danych lub układów peryferyjnych, co wiąże się z koniecznością wyprowadzenia na zewnątrz mikrokontrolera jego wewnętrznej magistrali danych i adresów, rolę tych szyn spełniają porty P0 i P2. Port P0 służy wówczas do przesyłania danych i młodszego bajtu słowa adresowego, zaś na porcie P2 wystawiany jest starszy bajt adresu.

Przy stosowaniu techniki multipleksowania danych i adresów niezbędny jest dodatkowy sygnał, umożliwiający rozróżnienie rodzaju informacji przesyłanej w danej chwili za pośrednictwem multipleksowanej szyny. W przypadku mikrokontrolerów rodziny '51 rolę tę spełnia sygnał ALE (*Address Latch Enable*), którego opadające zbocze

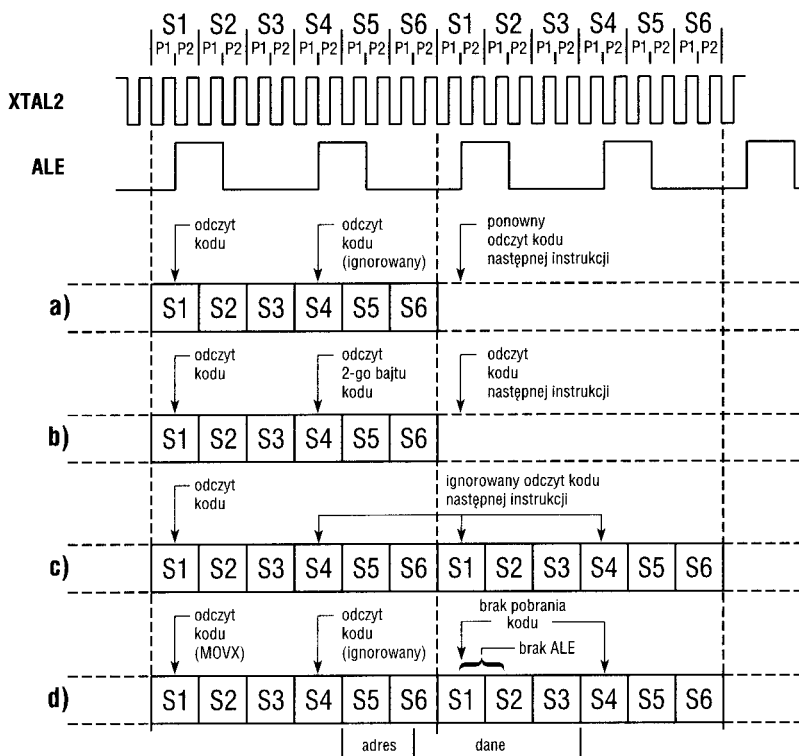
UWAGA

Przy współpracy z zewnętrzną pamięcią danych lub zewnętrznymi układami peryferyjnymi, do przesyłania informacji można wykorzystywać tryb adresowania 16-bitowego (z udziałem wskaźnika DPTR) – wówczas port P2 służy do wystawiania starszego bajtu adresu, bądź tryb adresowania 8-bitowego (z wykorzystaniem rejestrów R0, R1) – wtedy port P2 nie jest jawnie wykorzystywany do współpracy z układami zewnętrznymi i funkcjonuje jako zwykły port wejść/wyjść cyfrowych. Warto przy tym zauważyć, że jeśli przynajmniej część linii portu P2 służy do sterowania wejściami adresowymi układów zewnętrznych, to przy adresowaniu 16-bitowym na wejściach tych występują stany logiczne określone zawartością wskaźnika DPTR (a konkretnie jego bardziej znaczącej połówki – DPH), natomiast przy adresowaniu 8-bitowym stan tych wejść adresowych zależy od zawartości rejestru portu P2. Jeśli mikrokontroler korzysta z zewnętrznej pamięci programu, to bez względu na pojemność tej pamięci cały port P2 pełni rolę szyny adresowej – tym samym nie istnieje możliwość wykorzystania go jako zwykłego portu wejść/wyjść cyfrowych.

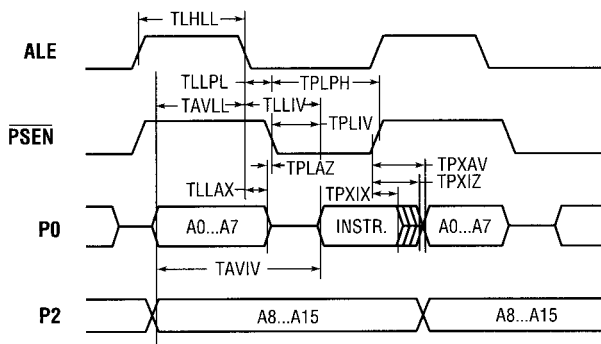
może być wykorzystane do zatrząśnięcia młodszego bajtu adresu (wystawionego na liniach portu P0) w dodatkowym rejestrze zewnętrznym. Sygnał ALE jest aktywowany dwukrotnie w każdym cyklu maszynowym, z wyjątkiem tych cykli, w których wykonywany jest rozkaz MOVX – sygnał ALE pojawia się wówczas tylko jeden raz. Tak więc, jeśli program nie zawiera rozkazu MOVX, częstotliwość sygnału ALE jest stała i równa 1/6 częstotliwości zegarowej mikrokontrolera. Sygnał ALE może być więc wówczas wykorzystany jako stabilne źródło sygnału taktującego dla urządzeń zewnętrznych.

Jeśli mikrokontroler współpracuje z zewnętrzną pamięcią programu, odczytem tej pamięci steruje sygnał PSEN. Sygnał PSEN jest aktywowany tylko podczas odczytów zewnętrznej pamięci programu – podczas pobierania rozkazów z wewnętrznej pamięci programu linia PSEN pozostaje nieaktywna. Podobnie jak w przypadku ALE, sygnał PSEN pojawia się dwukrotnie w ciągu cyklu maszynowego, chyba że wykonywany jest rozkaz MOVX – sygnał PSEN pojawia się wówczas tylko jeden raz. Linia PSEN jest aktywowana niezależnie od tego, czy odczytywana informacja jest potrzebna do wykonania bieżącej instrukcji, czy też nie (rys. 2.11).

W przypadku współpracy mikrokontrolera z zewnętrzną pamięcią danych (lub zewnętrznymi układami peryferyjnymi), sygnały zapisu WR i odczytu RD tej pamięci pojawiają się na liniach P3.6 i P3.7 portu P3 (są funkcjami alternatywnymi tych linii portu). Jak wspomniano wcześniej, w czasie pracy z zewnętrzną pamięcią programu wszystkie linie zarówno portu P0 jak i portu P2 pełnią funkcje alternatywne. Tak samo jest w przypadku gdy mikrokontroler wykonuje zapis lub odczyt zewnętrznej pamięci danych za pomocą rozkazu MOVX wykorzystującego wskaźnik DPTR. Jeśli natomiast mikrokontroler komunikuje się z zewnętrzną pamięcią danych stosując adresowanie pośrednie rejestrami R0 lub R1, to funkcje alternatywne pełnią tylko linie portu P0, podczas gdy stan linii portu P2 nie ulega zmianie. Przebiegi czasowe występujące podczas odczytu zewnętrznej pamięci programu oraz odczytu i zapisu zewnętrznej pamięci danych przedstawiono na rys. 2.12...2.14.



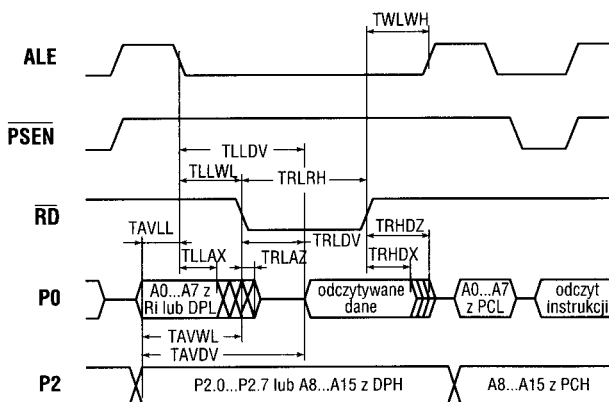
Rys. 2.11. Pobieranie kodu z pamięci programu w zależności od wykonywanej instrukcji:
a) instrukcja jednobajtowa wykonywana podczas jednego cyklu maszynowego (np. INC A);
b) instrukcja dwubajtowa wykonywana podczas jednego cyklu maszynowego (np. ADD A, #stała);
c) instrukcja dwubajtowa wykonywana podczas dwóch cykli maszynowych (np. INC DPTR);
d) instrukcja typu MOVX (dwa cykle maszynowe, kod jednobajtowy)



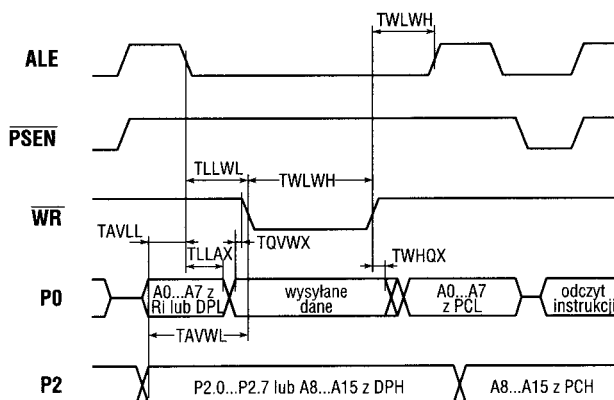
Rys. 2.12. Zależności czasowe dla cyklu odczytu zewnętrznej pamięci programu (wartości parametrów znajdują się w tab. 2.2)

Tab. 2.2. Orientacyjne wartości parametrów czasowych cykli odczytu zewnętrznej pamięci programu oraz odczytu i zapisu zewnętrznej pamięci danych. Wartości mogą ulegać nieznacznym zmianom w zależności od producenta i typu mikrokontrolera

Symbol	Parametr	Dla $f_{osc} = 12\text{MHz}$		Dla innych f_{osc}		Jednostki
		min.	maks.	min.	maks.	
1/t	Częstotliwość zegarowa			3,5	12,0	MHz
TLHLL	Długość impulsu ALE	127		2t – 40		ns
TAVLL	Adres ważny do ALE(L)	43		t – 40		ns
TLLAX	Adres ważny po ALE(L)	48		t – 35		ns
TLLIV	ALE(L) do instr. ważna		183		4t – 150	ns
TLLPL	ALE(L) do PSEN(L)	58		t – 25		ns
TPLPH	Długość impulsu PSEN	190		3t – 60		ns
TPLIV	PSEN(L) do instr. ważna		100		3t – 150	ns
TPXIX	PSEN(H) do instr. (3. stan)	0		0		ns
TPXIZ	PSEN(H) do instr. (3. stan)		63		t – 20	ns
TPXAV	PSEN(H) do adres ważny	75		t – 8		ns
TAVIV	Adres do instr. ważna		267		5t – 150	ns
TPLAZ	Adres (3. stan) do PSEN(L)		20		20	ns
TRLRH	Długość impulsu RD	400		6t – 100		ns
TWLWH	Długość impulsu WR	400		6t – 100		ns
TRLDV	RD (L) do dane ważne		252		5t – 165	ns
TRHDX	RD (H) do dane (3. stan)	0		0		ns
TRHDZ	RD (H) do dane (3. stan)		97		2t – 70	ns
TLLDV	ALE(L) do dane ważne		517		8t – 150	ns
TAVDV	Adres do dane ważne		585		9t – 165	ns
TLLWL	ALE(L) do RD (L) lub WR(L)	200	300	3t – 50	3t + 50	ns
TWHLH	RD (H) lub WR(H) do ALE(H)	33	133	t – 50	t + 50	ns
TAVWL	Adres do RD(L) lub WR(L)	203		4t – 130		ns
TQVWX	Dane ważne do zbocza	13		t – 70		ns
TQVWH	Dane ważne do WR(H)	433		7t – 150		ns
TWHQX	WR(H) do dane (3. stan)	33		t – 50		ns
TRLAZ	RD(L) do adres (3. stan)		20		20	ns



Rys. 2.13. Zależności czasowe dla cyklu odczytu zewnętrznej pamięci danych (wartości parametrów znajdują się w tab. 2.2)



Rys. 2.14. Zależności czasowe dla cyklu zapisu zewnętrznej pamięci danych (wartości parametrów znajdują się w tab. 2.2)

2.5.2. Instrukcje typu odczyt-modyfikacja-zapis

Budowa linii wejść/wyjść mikrokontrolerów rodziny '51 przedstawiona na rys. 2.9 wykazuje, że odczyt stanu portu może być przeprowadzony na dwa sposoby – jako odczyt stanu rejestru portu, bądź jako odczyt stanu linii wejściowych (wyprowadzeń) portu. Sposób wykonywania odczytu zależy od rodzaju instrukcji wykorzystanej do odczytu portu. Instrukcje, które odczytują zawartość rejestru (a nie wyprowadzeń) portu) określane są mianem instrukcji odczyt-modyfikacja-zapis, ponieważ wykonywanie tych instrukcji obejmuje odczyt zawartości rejestru portu, zmianę odczytanej wartości i zapis zmodyfikowanej wartości do rejestru portu. Instrukcjami odczyt-modyfikacja-zapis są następujące instrukcje:

ANL	(iloczyn logiczny, np. ANL P2, A),
ORL	(suma logiczna, np. ORL P1, #10),
XRL	(suma logiczna bitów modulo 2, np. XRL P3, A),
JBC	(skok jeśli bit jest jedynką i wyzerowanie bitu, np. JBC P1.3, DALEJ),
CPL	(negacja bitu, np. CPL P1.0),
INC	(inkrementacja bajtu, np. INC P2),
DEC	(dekrementacja bajtu, np. DEC P3),
DJNZ	(dekrementacja bajtu i skok jeśli nie zero, np. DJNZ P2, ALFA),
MOV Px.y, C	(przesłanie zawartości wskaźnika przeniesienia na bit y portu x, np. MOV P1.0, C),
CLR Px.y	(wyzerowanie bitu y portu x, np. CLR P1.1),
SETB Px.y	(ustawienie bitu y portu x, np. SETB P2.3).

Ostatnie trzy wymienione instrukcje są także (choć nie jest to wcale oczywiste) instrukcjami odczyt-modyfikacja-zapis, ponieważ ich wykonanie polega na odczycie stanu wszystkich 8 bitów rejestru portu, zmianie zawartości wybranego bitu i ponowny zapis tak zmodyfikowanego bajtu do rejestru portu. Wszystkie instrukcje odczytujące stan portu poza wymienionymi wyżej instrukcjami odczyt-modyfikacja-zapis, odczytują stan linii wejściowych (wyprowadzeń) portu.

Występowanie w liście instrukcji mikrokontrolerów rodziny '51 instrukcji, które w różny sposób interpretują stan portów pociąga za sobą istotne skutki, które łatwo można prześledzić na następującym przykładzie: Załóżmy, że linia P1.0 (linia 0 portu P1) wykorzystywana jest przez mikrokontroler jako linia wejściowa, linia P1.1 (linia 1 portu P1) steruje bezpośrednio bazą tranzystora bipolarnego, a pozostałe linie służą jako zwykłe wyjścia cyfrowe. Ponieważ linia P1.0 ma pracować jako wejście, bit 0 rejestru P1 musi być w stanie wysokim. Sterowanie tranzystorem za pomocą linii P1.1 polega na ustawianiu bitu 1 rejestru P1 – jeśli tranzystor ma przewodzić i zerowaniu bitu – jeśli tranzystor ma być wyłączony. Jeśli dodatkowo przyjmie się, że układ zewnętrzny wymusza na linii wejściowej P1.0 stan niski, a tranzystor jest włączony, to wykonanie instrukcji:

```
ANL P1, #0FFh      ; iloczyn logiczny zawartości portu P1
                   ; z bajtem złożonym z samych jedynek
```

nie pociągnie za sobą żadnej zmiany stanu portu P1. Wykonanie w tej samej sytuacji ciągu instrukcji:

```
MOV A, P1           ; przesłanie stanu portu P1 do akumulatora
ANL A, #0FFh        ; iloczyn logiczny akumulatora z bajtem
                   ; złożonym z samych jedynek
MOV P1, A           ; przesłanie zawartości akumulatora do
                   ; rejestru portu P1
```

które teoretycznie realizują (tyle że po kawałku) tę samą operację, co wymieniona wyżej instrukcja odczyt-modyfikacja-zapis, spowoduje zmianę stanu obu najmniej znaczących bitów rejestru portu P1. Zmiana ta wynika stąd, że instrukcja przesłania zawartości portu P1 do akumulatora wczyta stan linii portu, a nie stan rejestru portu P1, a oba wyprowadzenia P1.0 i P1.1 portu P1 są w stanie niskim (na P1.0 jest stan niski wymuszony z zewnątrz, a na P1.1 napięcie baza-emiter $U_{BE} \approx 0,7$ V włączonego tranzystora).

2.5.3. Obciążalność portów

Wszystkie linie wejść/wyjść mikrokontrolerów rodziny '51, z wyjątkiem linii portu P0, mogą być obciążone pojedynczym układem standardowej serii TTL (**tab. 2.3**). Obciążalność linii portu P0 jest dwukrotnie większa, jednakże jeśli port P0 realizuje funkcję zwykłych wyjść cyfrowych, stopnie końcowe linii portu pracują w układzie otwarty dren, co wymaga zwykle stosowania zewnętrznych rezystorów podciągających. Wszystkie linie portów pracujące jako wejścia mogą być sterowane przez układy TTL, NMOS lub CMOS. W przypadku portów innych niż P0 wejścia portów mogą być też sterowane bezpośrednio za pomocą układów typu otwarty kolektor lub otwarty dren. W obu tych przypadkach należy się jednak liczyć ze znacznym spowolnieniem przejść ze stanu niskiego do wysokiego, ponieważ jedynym elementem powodującym przejście linii portu ze stanu niskiego w stan wysoki jest rezystor podciągający zrealizowany na tranzystorze T2 (**rys. 2.10**).

Analizując obciążalność prądową linii portów warto zauważyć, że niektóre mikrokontrolery (np. C1051, C2051, C4051) mają znacznie podwyższoną wartość obciążalności w stanie niskim (maks. 20 mA przy $V_{CC} = 5$ V i 10 mA przy $V_{CC} = 2,7$ V). Można

Tab. 2.3. Orientacyjne wartości podstawowych parametrów stałoprądowych linii wejść/wyjść cyfrowych mikrokontrolerów rodziny '51 wykonywanych w technologii CMOS. Wartości podanych parametrów mogą ulegać nieznacznym zmianom w zależności od typu i producenta danego mikrokontrolera

Parametr	Oznaczenie	Wartości graniczne		Jednostki	Warunki pomiaru
		min.	maks.		
Napięcie wejściowe poziomu niskiego	V_{IL}	-0,5	$0,2V_{CC} - 0,1$	V	–
Napięcie wejściowe poziomu wysokiego	V_{IH}	$0,2V_{CC} + 0,9$	$V_{CC} + 0,5$	V	–
Napięcie wyjściowe poziomu niskiego dla linii portu P0, ALE i PSEN	V_{OL}	–	0,45	V	$I_{OL} = 3,2 \text{ mA}$
Napięcie wyjściowe poziomu niskiego dla pozostałych linii	V_{OL1}	–	0,45	V	$I_{OL} = 1,6 \text{ mA}$
Napięcie wyjściowe poziomu wysokiego dla linii portu P0, ALE i PSEN	V_{OH}	$2,4 \text{ V} \dots 0,9V_{CC}$	–	V	$I_{OH} = -800 \mu\text{A}$ $I_{OH} = -80 \mu\text{A}$
Napięcie wyjściowe poziomu wysokiego dla pozostałych linii	V_{OH1}	$2,4 \text{ V} \dots 0,9V_{CC}$	–	V	$I_{OH} = -80 \mu\text{A}$ $I_{OH} = -10 \mu\text{A}$
Prąd wejściowy dla stanu niskiego	I_{IL}	-10	-50	μA	$V_{IN} = 0,45 \text{ V}$
Prąd wejściowy przy przejściu ze stanu wysokiego do niskiego	I_{TL}	-65	-650	μA	$V_{IN} = 2,0 \text{ V}$

je zatem wykorzystać do bezpośredniego sterowania niektórych elementów (np. diod LED). Należy jednak zwrócić uwagę na fakt, iż producenci mikrokontrolerów określają również takie parametry jak np. maksymalna suma prądów wpływających do wyjść danego portu będących w stanie niskim, czy maksymalna moc tracona w układzie. Przy analizie pracy projektowanego układu należy uwzględnić wszystkie te ograniczenia, gdyż np. jednoczesne wymuszenie niskiego stanu (z prądem wpływającym 20 mA) na wszystkich ośmiu wyjściach portu P1 wyraźnie przekracza jeden z wtórnych parametrów obciążalności linii (sumę prądów) i niemal na pewno spowoduje uszkodzenie mikrokontrolera.

2.5.4. Taktowanie zewnętrznych urządzeń peryferyjnych

W projektowanym systemie występują niekiedy zewnętrzne urządzenia peryferyjne wymagające taktowania. W takiej sytuacji pierwszym nasuwającym się rozwiązaniem jest wykorzystanie tego samego sygnału do taktowania zarówno mikrokontrolera, jak i urządzeń peryferyjnych. Pomysł ten rzadko jednak daje się zrealizować w praktyce, z reguły bowiem częstotliwość taktowania mikrokontrolera wynosi kilka do kilkunastu MHz, podczas gdy częstotliwość taktowania wspomnianych urządzeń peryferyjnych powinna mieścić się w zakresie kilkuset kHz. W niektórych przypadkach jako sygnał taktujący można wykorzystać impulsy na wyprowadzeniach ALE lub PSEN. Częstotliwość pojawiania się tych impulsów nie zawsze jest jednak stała (patrz rozdz. 2.5.1), dlatego też znacznie wygodniejszym rozwiązaniem jest wytwarzanie sygnału

taktującego za pomocą układów CCU, PCA (występujących w niektórych bardziej rozbudowanych mikrokontrolerach rodziny '51) lub licznika T2 (nie wszystkie odmiany licznika T2 umożliwiają ten tryb pracy). W przypadkach, gdy sygnał taktujący zewnętrzne układy peryferyjne jest generowany za pomocą mikrokontrolera, należy uwzględnić skutki ewentualnego stosowania trybów pracy ze zmniejszonym poborem mocy. Przykładem takich efektów może być blokowanie sygnałów ALE i PSEN po przejściu mikrokontrolera w stan uśpienia lub zamrożenia.

**Wewnętrzne układy
peryferyjne
mikrokontrolerów
rodziny '51**



Wewnętrzne układy peryferyjne stanowią znaczącą, a z punktu widzenia konstruktora często najważniejszą część zasobów mikrokontrolera. Zastosowanie mikrokontrolera wyposażonego w rozbudowane wewnętrzne układy peryferyjne pozwala zwykle na istotne ograniczenie liczby i złożoności zewnętrznych układów peryferyjnych występujących w projektowanym systemie. Stąd też użycie mikrokontrolera droższego, ale odpowiednio dobranego do danego zastosowania (pod względem wewnętrznych zasobów), umożliwia z reguły obniżenie całkowitych kosztów konstruowanego urządzenia.

3.1. Układy licznikowe T0 i T1

Jednym z podstawowych rodzajów układów peryferyjnych umieszczanych wewnątrz mikrokontrolerów są układy licznikowe. W przypadku mikrokontrolerów rodziny '51 są to najczęściej układy licznikowe T0 i T1.

Mikrokontrolery rodziny '51 wyposażone są zwykle w co najmniej dwa układy licznikowe – liczniki T0 i T1 (jednym z nielicznych wyjątków jest układ C1051). Każdy z tych liczników składa się z dwóch 8-bitowych połówek. Połówki te widziane są przez CPU jako rejestry TH0 (bardziej znacząca część) i TL0 (mniej znacząca część) w przypadku licznika T0 oraz odpowiednio TH1 i TL1 w przypadku licznika T1. Wybór trybu pracy i sterowanie licznikami odbywa się za pomocą rejestrów TCON i TMOD. Każdy z obu liczników może pracować jako licznik (zliczanie impulsów zewnętrznych) lub jako czasomierz (zliczanie cykli maszynowych mikrokontrolera).

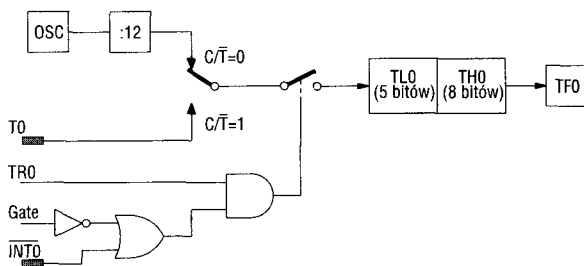
Przy pracy w trybie czasomierza inkrementacja licznika następuje w każdym kolejnym cyklu maszynowym, a zatem maksymalna częstotliwość zliczania jest równa $1/12$ częstotliwości zegarowej mikrokontrolera. W przypadku wykorzystywania układu w trybie licznika, jego zawartość jest zwiększana w odpowiedzi na opadające zbocze sygnału wejściowego. Detekcja zbocza odbywa się jednak synchronicznie z cyklem pracy mikrokontrolera – przez testowanie stanu odpowiedniej linii wejściowej w każdym kolejnym cyklu maszynowym (w stanie S3P1 cyklu). Jeśli testowanie wykazuje stan wysoki linii w jednym cyklu maszynowym oraz stan niski linii w następnym cyklu maszynowym, zawartość licznika jest zwiększana (podczas najbliższego stanu S3P1). Tak więc, aby zagwarantować wykrycie wszystkich impulsów, każdy (zarówno niski jak i wysoki) stan testowanej linii wejściowej musi trwać co najmniej jeden pełny cykl maszynowy. Skutkiem tego maksymalna częstotliwość pracy układu w trybie licznika jest ograniczona do $1/24$ częstotliwości zegarowej mikrokontrolera.

Niezależnie od rodzaju funkcji (czasomierz/licznik), każdy z układów licznikowych może działać w jednym z czterech, opisanych poniżej, trybów pracy.

Tryb 0

Tryb pracy 0 jest identyczny dla licznika T0 i licznika T1. W trybie tym układ pracuje jako 13-bitowy czasomierz/licznik. Stan licznika jest określany zawartością rejestru THn ($n=0$ lub 1) oraz pięciu młodszych bitów rejestru TLn. Stan trzech starszych bitów rejestru TLn jest nieokreślony. Przepelnienie licznika, czyli zliczenie impulsu, wskutek którego stan wszystkich bitów zmienia się z 1 na 0, ustawia jednocześnie bit TFn (rejestr TCON) sygnalizujący przerwanie od licznika.

Zliczanie ma miejsce tylko wtedy, gdy bit TRn jest w stanie wysokim oraz bądź $GATE=0$, bądź $\overline{INTn}=1$. Ustawienie bitu $GATE$ w stan wysoki pozwala na uruchamianie i blokowanie zliczania stanem linii \overline{INTn} , a tym samym na prowadzenie pomiarów czasu trwania impulsów zewnętrznych. Ustawienie bitu TRn (start licznika) nie powoduje zerowania rejestrów THn i TLn.

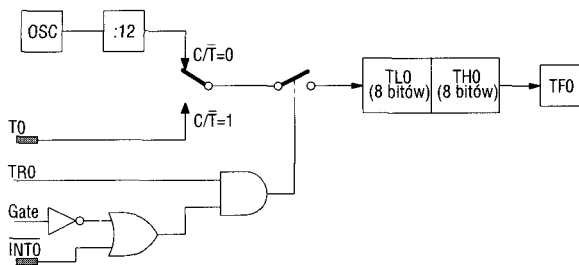


Rys. 3.1. Struktura licznika T0 w trybie pracy 0

Strukturę licznika w trybie pracy 0 pokazano na rys. 3.1 na przykładzie licznika T0. W przypadku licznika T1 symbole rejestrów i bitów sterujących będą występowały z indeksami 1 zamiast 0.

Tryb 1

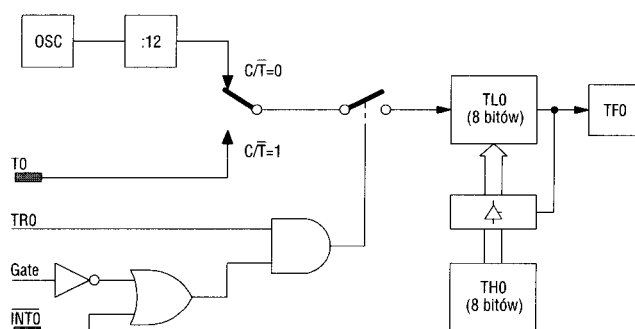
Tryb 1 pracy liczników T0 i T1 jest niemal identyczny z trybem 0. Jediną różnicą jest wykorzystywanie pełnej pojemności rejestrów TLn, a tym samym zwiększenie pojemności licznika z 13 do 16 bitów. Strukturę licznika w trybie 1 pracy pokazano na rys. 3.2 (na przykładzie licznika T0).



Rys. 3.2. Struktura licznika T0 w trybie pracy 1

Tryb 2

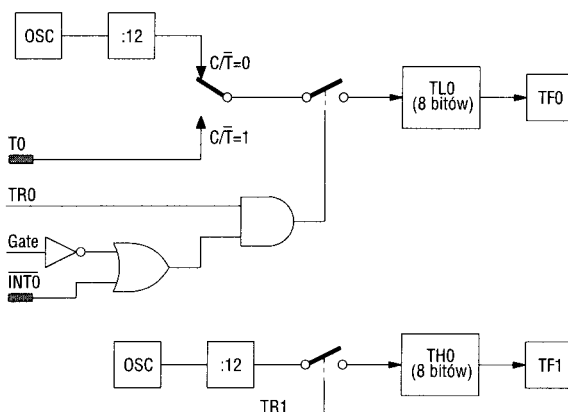
W trybie 2 liczniki pracują jako 8-bitowe liczniki (TLn) z automatycznym przeładowaniem (rys. 3.3). Przepełnienie z TLn nie tylko ustawia wskaźnik przerwania TFn, lecz jednocześnie ładuje TLn zawartością THn. Zawartość THn może być w dowolnej chwili zmieniana programowo. Operacja przeładowania nie powoduje zmiany zawartości rejestru THn.



Rys. 3.3. Struktura licznika T0 w trybie pracy 2

Tryb 3

Tryb 3 jest jedynym trybem, w którym sposób działania licznika T1 różni się od licznika T0. Wprowadzenie licznika T1 w tryb 3 pracy powoduje jego zatrzymanie, czyli daje ten sam efekt, co wyzerowanie bitu TR1 w którymkolwiek z pozostałych trybów pracy licznika T1. Licznik T0 w trybie 3 wykorzystuje swoje obydwie połówki (TH0 i TL0) jako dwa niezależne liczniki 8-bitowe (rys. 3.4). Jeden z nich (utworzony z TL0) funkcjonuje, podobnie jak cały licznik T0 w trybie 0 lub 1, przy czym jedyną różnicą jest pojemność licznika wynosząca 8 bitów. Druga część (TH0) tworzy 8-bitowy czasomierz zliczający cykle maszynowe. Włączanie i wyłączanie czasomierza odbywa się za pomocą bitu TR1, natomiast jego przepełnienie powoduje ustawienie wskaźnika przerwania TF1.



Rys. 3.4. Struktura licznika T0 w trybie pracy 3

Wykorzystywanie licznika T0 w trybie 3 pozbawia licznik T1 dwóch istotnych bitów sterujących - TR1 i TF1. Funkcję wyłączania licznika T1 można wprowadzić wtedy realizować wprowadzając go w jego własny tryb 3, jednakże brak własnego wskaźnika przerwania ogranicza możliwości zastosowań licznika T1 do takich, które nie wymagają stosowania przerwań. W szczególności licznik T1 może być wykorzystany do taktowania (określenia prędkości transmisji) łącza szeregowego.

Wewnętrzne zasoby mikrokontrolerów o zmniejszonej liczbie wyprowadzeń są nieco zmodyfikowane w stosunku do zasobów mikrokontrolera C51. W szczególności dotyczy to mikrokontrolera C1051 firmy Atmel, w którym nie występuje licznik T1.

3.2. Układ licznikowy T2

W bardzo krótkim czasie po opracowaniu mikrokontrolera 8051 okazało się, że w wielu zastosowaniach zachodzi potrzeba wykorzystania większej liczby układów licznikowych. Dlatego też większość opracowanych później mikrokontrolerów wyposażono w dodatkowe układy licznikowe. Pierwszym z takich mikrokontrolerów był układ 8052. Warto zwrócić uwagę na fakt, że do niedawna układ licznikowy T2 występował wyłącznie w układach pochodnych 8052/C52 i bardziej rozbudowanych, jednak ostatnio firma Philips zaczęła wyposażać w ten układ także mikrokontrolery C51. Jest to doskonały przykład na to, jak duże różnice mogą kryć się w na pozór identycznych (a w każdym razie analogicznie oznaczanych) mikrokontrolerach różnych producentów.

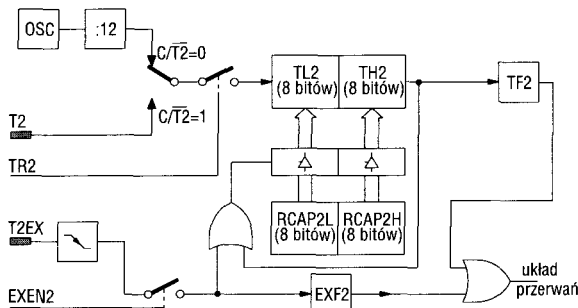
3.2.1. Licznik T2 mikrokontrolera 8052

Układ licznikowy T2 mikrokontrolera 8052 może, podobnie jak układy licznikowe T0 i T1, pracować jako licznik (zliczanie impulsów wejściowych z wyprowadzenia T2 mikrokontrolera), bądź jako układ czasowy (zliczanie cykli maszynowych). Sposób działania układu określony jest stanem bitu $C/\overline{T2}$ umieszczonego w rejestrze T2CON. Układ licznikowy T2 może pracować w jednym z następujących trybów:

- praca z automatycznym przeładowywaniem (*auto-reload*),
- praca z przechwytywaniem (*capture*),
- generator sygnału taktującego dla portu szeregowego.

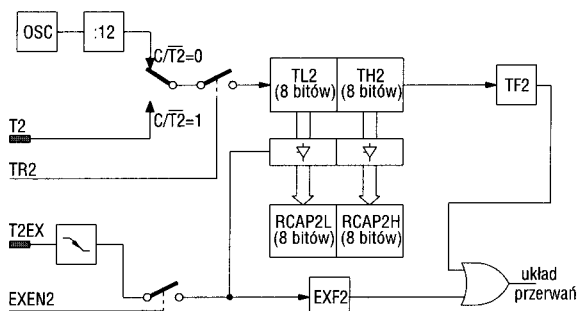
Tryb pracy układu licznikowego definiowany jest stanem bitów RCLK, TCLK, $CP/\overline{RL2}$ oraz TR2, znajdujących się w rejestrze T2CON. We wszystkich trybach układ licznikowy T2 pracuje jako układ 16-bitowy, złożony z dwóch części 8-bitowych, widzianych przez CPU jako rejestry TL2 (mniej znaczące 8 bitów układu) i TH2 (bardziej znaczące 8 bitów).

Podczas pracy w trybie z automatycznym przeładowywaniem (rys. 3.5) przepełnienie licznika (układ licznika T2 zlicza wyłącznie w górę) powoduje ustawienie wskaźnika



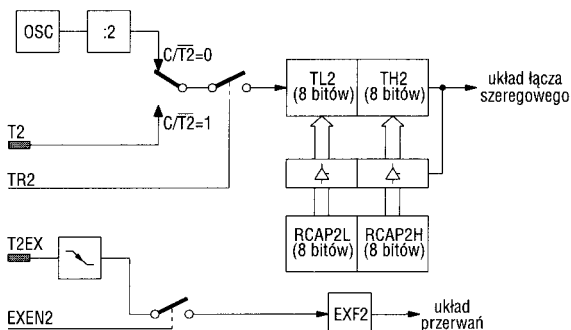
Rys. 3.5. Struktura licznika T2 podczas pracy w trybie z automatycznym przeładowywaniem

przerwania TF2, znajdującego się w rejestrze T2CON i jednocześnie przeładowanie układu licznikowego zawartością rejestrów RCAP2L (mniej znaczący bajt) i RCAP2H (bardziej znaczący bajt). Jeśli bit EXEN2 w rejestrze T2CON jest ustawiony, to przeładowanie licznika zawartością rejestrów RCAP2L i RCAP2H będzie następowało także wskutek opadającego zbocza na wyprowadzeniu T2EX mikrokontrolera. Zbocze to ustawi jednocześnie wskaźnik przerwania EXF2 (rejestr T2CON). Przeładowywanie licznika nie powoduje zmiany zawartości rejestrów RCAP2L i RCAP2H.



Rys. 3.6. Struktura licznika T2 w trybie pracy z przechwytywaniem

W trybie pracy z przechwytywaniem (rys. 3.6), przepełnienie się licznika powoduje, podobnie jak w trybie z automatycznym przeładowaniem, ustawienie wskaźnika przerwania TF2. Jeśli dodatkowo bit EXEN2 jest ustawiony, to pojawienie się opadającego zbocza na wyprowadzeniu T2EX mikrokontrolera spowoduje przepisanie zawartości licznika do rejestrów RCAP2L i RCAP2H, z jednoczesnym ustawieniem wskaźnika przerwania EXF2.



Rys. 3.7. Struktura licznika T2 podczas pracy jako generator prędkości transmisji łącza szeregowego

Konfigurację układu licznikowego T2 pracującego jako generator prędkości transmisji dla łącza szeregowego przedstawiono na rys. 3.7. Przejście układu licznikowego w ten tryb pracy następuje wskutek ustawienia bitu TCLK, RCLK lub obydwu tych bitów. Ustawienie tylko jednego z tych bitów, przy pozostawieniu drugiego z nich w stanie niskim, umożliwia uzyskanie innej prędkości transmisji dla nadawania, a innej dla odbioru danych przesyłanych łączem szeregowym. W omawianym trybie pracy konfiguracja układu licznikowego jest zbliżona do konfiguracji układu w trybie z automatycznym przeładowywaniem. Istotną różnicą jest jednak niewykorzystywanie wskaźni-



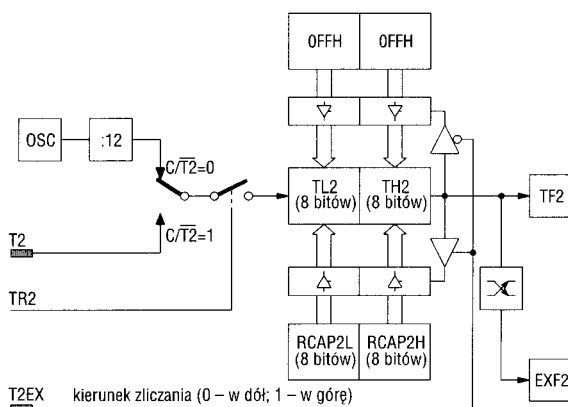
W niektórych mikrokontrolerach wyposażonych w układ licznikowy T2 funkcję bitów RCLK i TCLK spełnia jeden wspólny bit RTCLK – patrz struktura rejestru T2CON w różnych mikrokontrolerach.

ka przerwania TF2 oraz odseparowanie układu reakcji na opadające zbocze sygnału z wyprowadzenia T2EX mikrokontrolera. Wyprowadzenie T2EX może natomiast być wykorzystane jako wejście zewnętrznego przerwania ustawiającego wskaźnik EXF2 ujemnym zboczem.

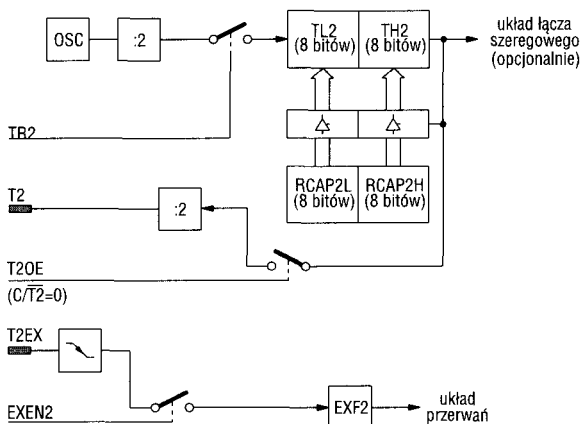
W przypadku pracy układu licznikowego T2 w konfiguracji czasomierza jako generatora prędkości transmisji łączy szeregowego, współczynnik podziału częstotliwości zegarowej mikrokontrolera jest równy 2, a nie 12, jak w pozostałych trybach pracy układu. W związku z tym na każdy cykl maszynowy przypada 6 zliczonych impulsów. W konsekwencji próby odczytu lub zapisu rejestrów TH2 i TL2 mogą dawać wynik nieokreślony. Z tego też powodu zapis do rejestrów RCAP2L i RCAP2H powinien być wykonywany przy wyłączonym zliczaniu (bit TR2 równy zeru).

3.2.2. Modyfikacje układu licznikowego T2

Niektóre mikrokontrolery rodziny '51 mają bardziej rozbudowany układ licznikowy T2. Spotyka się dwa rodzaje modyfikacji tego licznika. Pierwszą z nich jest możliwość zliczania zarówno w górę, jak i w dół podczas pracy w trybie z automatycznym przeładowywaniem (rys. 3.8). W celu wprowadzenia licznika w ten tryb pracy należy ustawić bit DCEN umieszczony jest w rejestrze T2MOD (rejestr ten nie występuje w mikrokontrolerze 8052). Kierunek zliczania (w górę/w dół) zależy od stanu wyprowadzenia T2EX (stan niski – zliczanie w dół; stan wysoki – zliczanie w górę). Przy zliczaniu w górę przepełnienie licznika, a tym samym ustawienie wskaźnika przerwania od przepełnienia licznika T2 oraz przeładowanie licznika zawartością rejestrów RCAP2H i RCAP2L następuje w chwili zliczenia kolejnego impulsu przy stanie licznika 0FFFFh. Jeśli zliczanie odbywa się w dół, to ustawienie wskaźnika



Rys. 3.8. Struktura zmodyfikowanego licznika T2 z możliwością zliczania w górę i w dół w trybie z automatycznym przeładowywaniem



Rys. 3.9. Struktura licznika T2 w trybie pracy jako generatora fali prostokątnej

przerwania od przepełnienia licznika T2 następuje w chwili zrównania się zawartości rejestrów RCAP2H i RCAP2L z zawartością bajtów TH2 i TL2 licznika. Oprócz ustawienia wskaźnika przerwania następuje wówczas przeładowanie licznika wartością 0FFFFh. W omawianym trybie pracy licznika T2 każde przepełnienie (zarówno podczas zliczania w górę, jak i w dół) powoduje zmianę stanu bitu EXF2. Bit EXF2 może być więc traktowany w tym trybie pracy licznika jako jego 17 bit.

Część mikrokontrolerów umożliwia wykorzystanie licznika T2 do generowania na wyprowadzeniu T2 (linia P1.0) mikrokontrolera fali prostokątnej o programowalnej częstotliwości i wypełnieniu 50% (rys. 3.9). Przejście w ten tryb pracy licznika T2 następuje wskutek ustawienia bitu T2OE (umieszczonego w rejestrze T2MOD) i wyzerowania bitu C/T2 (w rejestrze T2CON). Przy pracy w trybie generowania fali prostokątnej przepełnienie licznika T2 nie powoduje ustawienia wskaźnika przerwania od przepełnienia licznika. Częstotliwość generowanej fali prostokątnej zależy od częstotliwości taktowania mikrokontrolera oraz stanu rejestrów RCAP2H i RCAP2L. Traktując zawartość rejestrów RCAP2H i RCAP2L jako odpowiednio starszy i młodszy bajt 16-bitowej liczby RCAP2, częstotliwość generowanej fali prostokątnej można określić wzorem:

$$f_{\text{out}} = \frac{f_{\text{OSC}}}{4 \cdot (65536 - \text{RCAP2})}$$

W przypadku niektórych mikrokontrolerów firmy Intel, przeznaczonych do pracy z częstotliwością taktowania do 20 MHz, maksymalna częstotliwość generowanej fali prostokątnej jest ograniczona do wartości mniejszej niż wynikałoby to z powyższego wzoru. Omawiane wersje mikrokontrolerów nie mogą generować fali o częstotliwości wyższej niż 4 MHz. Oznacza to, że przy częstotliwości taktowania powyżej 16 MHz liczba RCAP2 może przyjmować co najwyżej wartość 0FFFEh.

Układ licznikowy T2 może być wykorzystywany jednocześnie do generowania fali prostokątnej oraz do taktowania układu łącza szeregowego. Częstotliwość generowanej fali oraz prędkość transmisji nie mogą być jednak w tym przypadku ustalane niezależnie, ponieważ obie zależą od zawartości rejestrów RCAP2H i RCAP2L.

3.3. Interfejsy szeregowo

Znaczna część, jeśli nie zdecydowana większość, mikrokontrolerów rodziny '51 jest wyposażona w co najmniej jeden sprzętowy interfejs szeregowy umożliwiający transmisję asynchroniczną. Dzięki niemu można łatwo wymieniać dane z otoczeniem za pomocą interfejsu RS 232, a urządzenie skonstruowane z wykorzystaniem mikrokontrolera może być połączone z komputerem nadrzędnym (np. typu IBM PC) lub dowolnym urządzeniem peryferyjnym (np. drukarką) mającym taki interfejs. Prędkość transmisji łącza może być przy tym w szerokim zakresie zmieniana programowo – uzyskanie standardowych prędkości transmisji nie stanowi zazwyczaj problemu.

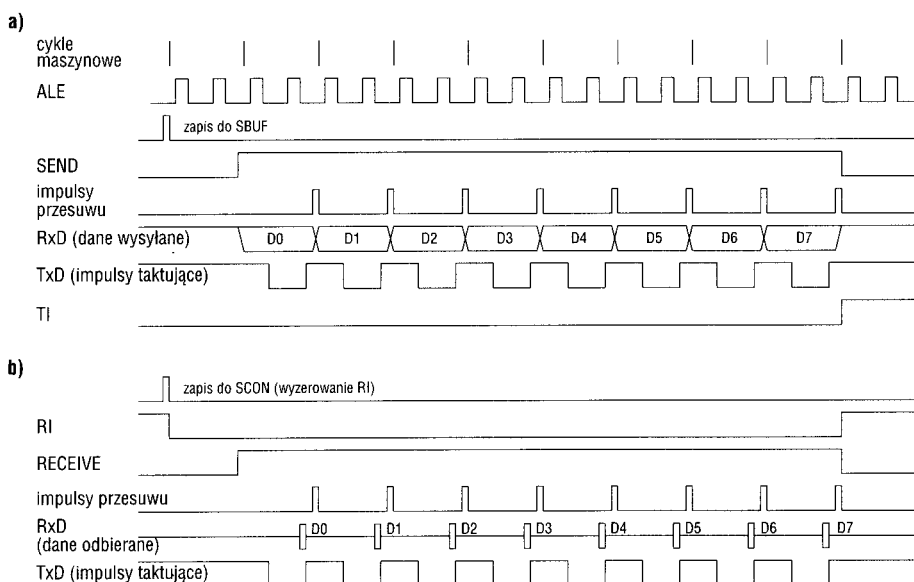
3.3.1. Standardowe łącze szeregowo

Standardowe łącze szeregowo występujące w mikrokontrolerach rodziny '51 umożliwia prowadzenie synchronicznej, bądź asynchronicznej transmisji danych, przy czym w przypadku pracy asynchronicznej nadawanie i odbiór danych są całkowicie niezależne (*full-duplex*). Układ odbiornika łącza szeregowego jest wyposażony w bufor danych. Pozwala to na rozpoczęcie odbierania następnego bajtu danych przed odczytaniem z rejestru odbiornika SBUF poprzednio odebranego bajtu danych. Omawiany bufor danych ma jednak pojemność tylko jednego bajtu. Jeśli zatem odczyt rejestru SBUF nie nastąpi do momentu zakończenia kolejnej transmisji, to informacja z poprzedniego bajtu danych zostanie nieodwracalnie utracona.

Podczas transmisji asynchronicznej nadawane dane wysyłane są za pomocą linii TxD, zaś odbiór danych następuje z wyprowadzenia RxD mikrokontrolera. W przypadku transmisji synchronicznej zarówno nadawane, jak i odbierane dane przesyłane są po linii RxD, podczas gdy wyprowadzenie TxD funkcjonuje jako wyjście sygnału taktującego. Bajty danych są transmitowane zawsze począwszy od bitu najmniej znaczącego. Interfejs USART może pracować w jednym z czterech niżej opisanych trybów.

Tryb 0

W trybie 0 transmisja jest synchroniczna, przesyłanych jest 8 bitów danych, a prędkość transmisji jest stała i równa 1/12 częstotliwości zegarowej mikrokontrolera. Nadawanie danych jest inicjalizowane przez zapis bajtu przeznaczonego do wysłania do rejestru SBUF. Zapis ten następuje w stanie S6P2, ale nadawanie pierwszego bitu rozpoczyna się dopiero w stanie S6P2 następnego cyklu maszynowego (czyli po upływie jednego pełnego cyklu) – rys. 3.10a. W tym właśnie momencie jest ustawiany wewnętrzny bit stanu nadajnika SEND, a wysoki stan tego bitu uaktywnia alternatywne funkcje typu wyjście linii P3.1 (linia danych) i P3.0 (linia sygnału taktującego). Przed rozpoczęciem i po zakończeniu nadawania linia sygnału taktującego jest utrzymywana w stanie wysokim. Podczas nadawania (w czasie gdy bit SEND ma wartość jedynki logicznej) linia ta jest utrzymywana na poziomie niskim podczas stanów S3, S4 i S5, na poziomie wysokim zaś w stanach S6, S1 i S2 kolejnych cykli maszynowych. Kolejne bity danych pojawiają się na linii P3.1 w stanie S6P2. W stanie S1P1 dziesiątego cyklu maszynowego (licząc od chwili zapisu do rejestru SBUF) bit SEND jest zerowany, a jednocześnie ustawiany jest wskaźnik przerwania od nadajnika łącza szeregowego (bit TI).



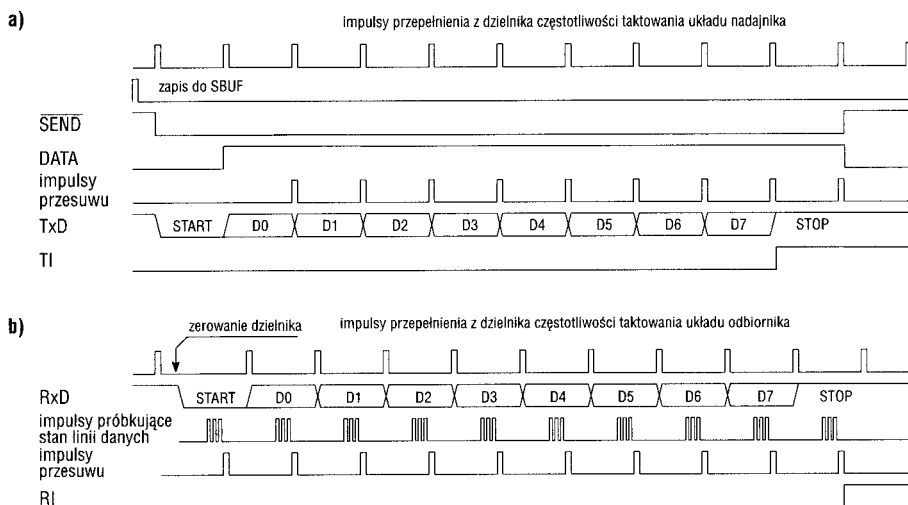
Rys. 3.10. Zależności czasowe operacji nadawania (a) i odbioru (b) podczas pracy łącza szeregowego w trybie 0

Odbiór danych jest inicjalizowany przez wymuszenie $REN=1$ i $RI=0$ (oba bity znajdują się w rejestrze SCON). W stanie S6P2 następnego cyklu maszynowego następuje ustawienie wewnętrznego bitu stanu odbiornika RECEIVE, który uaktywnia alternatywną funkcję typu wyjście linii P3.1. Stan tej linii zmienia się w stanach S3P1 i S6P1, zaś stan linii P3.0 jest próbkowany w stanach S5P2 kolejnych cykli maszynowych (rys. 3.10b). W stanie S1P1 dziesiątego cyklu maszynowego zerowany jest bit RECEIVE i ustawiany jest wskaźnik przerwania od odbiornika łącza szeregowego (bit RI). Odebrany bajt danych przepisywany jest do rejestru SBUF w stanie S6P2 ostatniego cyklu maszynowego, podczas którego $RECEIVE=1$.

Tryb 1

Transmisja w trybie 1 jest asynchroniczna – przesyłany jest bit startu (zero), 8 bitów danych i bit stopu (jedyńska). Przy odbiorze bit stopu jest umieszczany w RB8 (w rejestrze SCON). Prędkość transmisji w trybie 1 może być zmieniana w szerokim zakresie. Sposób programowania prędkości transmisji opisano dalej (rozdz. 3.3.2).

Sygnał taktujący wykorzystywany przez układ nadajnika i odbiornika pracującego w trybie asynchronicznym jest podawany na 4-bitowy licznik (oddzielny dla nadajnika i odbiornika). W przypadku nadajnika licznik jest wykorzystywany jako zwykły dzielnik przez 16. Nadawanie bajtu danych jest wyzwalane zapisem do rejestru SBUF i rozpoczyna się w stanie S1P1 występującym po przepełnieniu dzielnika (rys. 3.11a). Nadawanie jest zatem synchronizowane z pracą dzielnika (a nie z zapisem do rejestru SBUF). We wspomnianym stanie S1P1 jest zerowany wewnętrzny bit stanu nadajnika \overline{SEND} , a na linii TxD jest wymuszany stan niski (bit startu). Kolejne przepełnienie dzielnika powoduje ustawienie wewnętrznego bitu stanu DATA i wymuszenie na linii



Rys. 3.11. Zależności czasowe operacji nadawania (a) i odbioru (b) podczas pracy interfejsu szeregowego w trybie 1

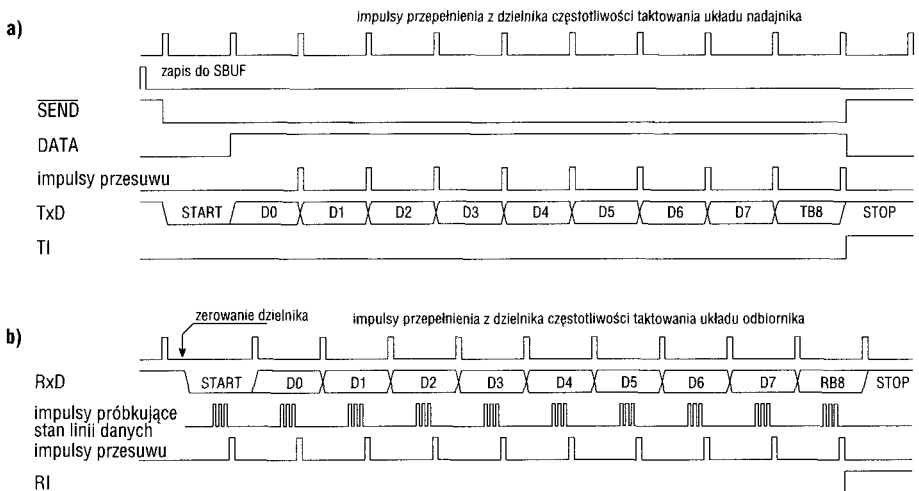
TxD stanu odpowiadającego wartości pierwszego z nadawanych bitów danych. Każde następne przepełnienie dzielnika powoduje zmianę stanu linii TxD zgodnie z zawartością kolejnych bitów nadawanego bajtu danych. Dziewiąte przepełnienie dzielnika powoduje ustawienie wskaźnika przerwania TI i wymuszenie na linii TxD stanu wysokiego (bit stopu), dziesiąte zaś zeruje bit DATA i ustawia bit stanu $\overline{\text{SEND}}$, kończąc tym samym operację nadawania.

Operacja odbierania bajtu danych jest wyzwalana w wyniku wykrycia opadającego zbocza na linii RxD (rys. 3.11b). W tym celu stan linii RxD jest testowany z częstotliwością impulsów wchodzących na 4-bitowy licznik współpracujący z układem odbiornika, czyli 16-krotnie częściej niżby wynikało to z ustawionej prędkości transmisji. W chwili wykrycia opadającego zbocza sygnału RxD licznik jest zerowany. Dzięki temu kolejne przepełnienia licznika będą wypadały na przełomie (koniec/początek) kolejnych odbieranych bitów. Uwzględniając stan licznika, czas trwania każdego bitu można podzielić na 16 odcinków. Stan linii RxD jest testowany trzykrotnie w ciągu każdego bitu – w chwili, gdy licznik przyjmuje wartość 7, 8 i 9. Za wczytany przyjmuje się stan, jaki wystąpił co najmniej dwa razy podczas trzykrotnego próbkowania linii RxD – jest to swoisty mechanizm filtracji krótkich zakłóceń. Jeśli określona w taki sposób wartość pierwszego odebranego bitu jest różna od zera, to cały układ odbiornika jest zerowany i powraca do oczekiwania na opadające zbocze sygnału RxD. W ten sposób eliminowane są fałszywe bity startu, stanowiące efekt silnego, krótkiego zakłócenia na linii RxD. Jeśli odebrany bit startu jest prawidłowy (równy zeru), to procedura odbioru danych jest kontynuowana (wczytywane są kolejne bity) aż do odbioru bitu stopu włącznie. Przepisanie odebranego bajtu danych do rejestru SBUF, wartości bitu stopu do RB8 (rejestr SCON) i ustawienie wskaźnika przerwania RI następuje w momencie odebrania 10 bitu (bitu stopu). Warunkiem wykonania wymienionych trzech operacji jest jednak RI=0 i albo SM2=0, albo ważny (równy jeden) bit stopu – w przeciwnym razie odebrane dane są nieodwracalnie tracone. Bez

względem na to, czy powyższe warunki zostały spełnione, czy też nie, układ odbiornika wraca do oczekiwania na kolejne opadające zbocze sygnału RxD.

Tryb 2

W trybie 2 pracy łącza szeregowego transmisja jest asynchroniczna, przesyłany jest bit startu (zero), 9 bitów danych i bit stopu (jedynek). Przy nadawaniu 9. bitem danych jest bit TB8 z rejestru SCON. Stan bitu TB8 może być zmieniany programowo, tak więc nadanie mu odpowiedniej wartości pozwala na realizowanie transmisji z bitem parzystości/nieparzystości, dodatkowym bitem stopu itp. Przy odbiorze 9. bit odbieranych danych jest umieszczany w RB8 (w rejestrze SCON), natomiast bit stopu jest ignorowany. Prędkość transmisji może być równa 1/32 lub 1/64 częstotliwości zegarowej mikrokontrolera.



Rys. 3.12. Zależności czasowe operacji nadawania (a) i odbioru (b) podczas pracy interfejsu szeregowego w trybie 2 lub 3

Praca układów nadajnika i odbiornika łącza szeregowego w trybie 2 jest prawie taka sama, jak w trybie 1. Różnice wynikają jedynie ze zwiększenia liczby przesyłanych bitów danych (rys. 3.12). W przypadku nadawania wskaźnik TI jest ustawiany o jeden bit później (przy dziesiątym przepełnieniu dzielnika), czyli równocześnie z wyzerowaniem bitu DATA i ustawieniem bitu SEND. W przypadku odbioru przepisanie odebranego bajtu danych do rejestru SBUF, wartości 9. bitu danych do RB8 i ustawienie wskaźnika przerwania RI następuje w momencie odebrania 9. bitu danych. Warunkiem wykonania wymienionych trzech operacji jest RI=0 i albo SM2=0, albo równy jeden 9. bit danych.

Tryb 3

Tryb 3 jest niemal identyczny z trybem 2 pracy łącza szeregowego. Jedyną różnicą dotyczy, opisanego w rozdz. 3.3.2, programowania prędkości transmisji. Wybór trybu pracy łącza szeregowego dokonywany jest przez nadanie odpowiednich wartości bitom SM0 i SM1 umieszczonym w rejestrze SCON. We wszystkich czterech trybach pracy łącza nadawanie danych inicjowane jest przesłaniem bajtu danych do rejestru SBUF.



Programowo zarówno do nadawania, jak i do odbioru wykorzystuje się ten sam adres w przestrzeni SFR (rejestr SBUF). Fizycznie adres ten jest jednak wykorzystany przez dwa niezależne rejestry: jeden - do zapisu danych do nadawania, drugi - zawierający dane odebrane. Nie zachodzi zatem niebezpieczeństwo kolizji danych do nadawania umieszczanych w rejestrze SBUF, z danymi odbieranymi, które odczytywane są także z rejestru SBUF.

W trybie 0 odbiór danych rozpoczyna się w momencie spełnienia warunków RI=0 i REN=1 (oba bity znajdują się w rejestrze SCON). W pozostałych trybach odbiór bajtu danych jest zapoczątkowany pojawieniem się bitu startu, przy jednoczesnym spełnieniu warunku REN=1. Odebrany bajt jest umieszczany w rejestrze SBUF.

Wysłanie lub odebranie pełnej ramki (kompletu bitów: startu, danych i stopu) powoduje ustawienie umieszczonych w rejestrze SCON wskaźników przerwań od nadajnika (bit TI) lub odbiornika (bit RI). Oba przerwania mają ten sam wektor adresowy, tak więc źródło przerwania (nadajnik/odbiornik) musi być określone programowo przez procedurę obsługi przerwania. Oba wskaźniki przerwań (TI i RI) muszą być zerowane programowo. Bitem zezwolenia na przerwanie od łącza szeregowego jest bit ES (w rejestrze IE).

3.3.2. Programowanie prędkości transmisji interfejsu

Prędkość transmisji danych po łączu szeregowym zależy w znacznej mierze od wykorzystywanego trybu pracy łącza. W trybie 0 prędkość transmisji jest stała i wynosi:

$$BR_0 = \frac{f_{OSC}}{12}$$

gdzie f_{OSC} jest częstotliwością taktowania mikrokontrolera. W trybie 2 prędkość transmisji zależy od wartości bitu SMOD (rejestr PCON) i jest równa:

$$BR_2 = \frac{f_{OSC} \cdot 2^{SMOD}}{64}$$

W trybach 1 i 3 pracy łącza szeregowego prędkość transmisji jest ustalana przez częstotliwość przepełniania się licznika T1 lub licznika T2 (w mikrokontrolerze 8052 lub innych, które mają taki układ licznikowy). W niektórych mikrokontrolerach (np. C517) można też do tego celu wykorzystać specjalizowany układ generatora prędkości transmisji. W przypadku wykorzystania licznika T1 do taktowania układu łącza szeregowego, prędkość transmisji jest określona jako:

$$BR_{1,3} = \frac{2^{SMOD} \cdot \text{częstotliwość przepełniania się licznika T1}}{32}$$

Licznik T1 może być wykorzystywany w dowolnym trybie pracy, jednak najczęściej wykorzystuje się tryb pracy z automatycznym przeładowywaniem (tryb 2). Prędkość transmisji łącza szeregowego jest wówczas określona wzorem:

$$BR_{1,3} = \frac{f_{OSC} \cdot 2^{SMOD}}{32 \cdot 12 \cdot (256 - TH1)}$$

W celu uzyskania bardzo małych prędkości transmisji można wykorzystać tryb 1 pracy licznika T1 (16-bitowy czasomierz), jednakże wówczas operację przeładowywania zawartości licznika trzeba wykonywać programowo – np. w odpowiedzi na przerwanie od przepełnienia licznika T1. W przypadku wykorzystania licznika T1 pracującego w trybie 2 operacja przeładowywania jest realizowana w pełni sprzętowo i przerwanie od licznika T1 może pozostać niewykorzystane.

W mikrokontrolerze 8052 (i innych zawierających analogiczny układ licznika T2) do wytwarzania sygnału taktującego układ łącza szeregowego można wykorzystać licznik T2. W tym celu bity TCLK i/lub RCLK w rejestrze T2CON muszą być ustawione. Zastosowanie oddzielnych bitów sterujących dla nadajnika (TCLK) i odbiornika (RCLK) umożliwia uzyskanie innej prędkości pracy łącza przy nadawaniu, a innej przy odbiorze, ponieważ jedna z nich może być określona za pomocą licznika T1, a druga przy użyciu licznika T2. Prędkość transmisji łącza szeregowego taktowanego za pomocą licznika T2 wynosi:

$$BR_{1,3} = \frac{f_{OSC}}{32 \cdot (65536 - RCAP2)}$$

gdzie RCAP2 jest 16-bitową liczbą całkowitą bez znaku, umieszczoną w parze rejestrów RCAP2H (starszy bajt) i RCAP2L (młodszy bajt).

3.3.3. Przesyłanie informacji łączem szeregowym w systemach wieloprocesorowych

Zastosowane w mikrokontrolerach rodziny '51 łącze szeregowe ma mechanizmy umożliwiające łatwe przystosowanie łącza do przesyłania danych w systemach wieloprocesorowych. Mechanizmy te są uaktywniane przez ustawienie bitu SM2 znajdującego się w rejestrze SCON. Ustawienie bitu SM2 powoduje, że podczas pracy łącza w trybie 2 lub 3, dane odbierane za pośrednictwem łącza szeregowego są akceptowane (przesyłane do rejestru odbiornika SBUF z jednoczesnym ustawieniem wskaźnika przerwania RI) tylko wtedy, gdy 9. bit odebranych danych (umieszczany w RB8) jest jedynką. W przeciwnym razie odebrana ramka jest ignorowana. Wykorzystując tę właściwość odbiornika, wymianę informacji po łączu szeregowym w systemie wieloprocesorowym, w którym jeden z mikrokontrolerów pełni rolę procesora nadrzędnego, można zrealizować na przykład następująco:

- Ustawić bity SM2 we wszystkich procesorach podrzędnych i wprowadzić łącza szeregowo do pracy w trybie 2 lub 3 (np. podczas inicjalizacji pracy mikrokontrolerów).
- Wysłać (z procesora nadrzędnego) adres procesora podrzędnego, dla którego przeznaczona jest najbliższa transmisja danych. Adres należy zakodować na 8 bitach, a 9. wysyłany bit ustawić.
- Procedura obsługi przerwania od odbiornika łącza szeregowego procesorów podrzędnych musi przeanalizować odebrany adres, by wyzerować bit SM2, jeśli dany procesor został zaadresowany.
- Przeprowadzić jedno- lub dwukierunkową transmisję między procesorem nadrzędnym a zaadresowanym uprzednio procesorem podrzędnym. Wszystkie przesyłane łączem dane powinny mieć 9. bit wyzerowany. Tym sposobem dane te zostaną

zignorowane przez wszystkie pozostałe (niezaadresowane) procesory podrzędne, gdyż ich bit SM2 jest ustawiony.

- e) Po zakończeniu transmisji rozadresować procesor podrzędny ustawiając jego bit SM2. System będzie trwał w oczekiwaniu na ponowne wykonanie operacji z punktu b).

Bit SM2 w trybie 1 pracy łącza szeregowo można wykorzystać do testowania poprawności bitu stopu – ustawiony bit SM2 będzie powodował ignorowanie odebranych danych w przypadku nieprawidłowego bitu stopu. W przypadku pracy łącza w trybie 0 stan bitu SM2 jest zwykle bez znaczenia dla pracy łącza, niektórzy producenci mikrokontrolerów zalecają jednak zerowanie bitu SM2 przy pracy łącza w trybie 0.

3.3.4. Rozszerzone łącze szeregowo

Podstawowy układ łącza szeregowo może być rozbudowany tak, by pełnił dodatkowe funkcje – sygnalizował błędny bit stopu i automatycznie rozpoznawał adresy (w komunikacji wieloprocesorowej). Tak rozbudowany układ łącza szeregowo określany jest mianem rozszerzonego łącza szeregowo i występuje m.in. w mikrokontrolerach C52 firmy Intel oraz najnowszych wersjach układu C51 produkowanego przez firmę Philips.

Sygnalizacja braku bitu stopu w mikrokontrolerach z rozszerzonym łączem szeregowym polega na ustawieniu bitu FE, który może być odczytany za pośrednictwem rejestru SCON (bit SM0/FE).

Automatyczne rozpoznawanie adresów ma zastosowanie głównie w komunikacji wieloprocesorowej, gdzie każdy pakiet danych jest poprzedzany adresem mikrokontrolera, dla którego przeznaczone są przesyłane dane. Każdy taki bajt adresu odebrany przez port szeregowo wymaga normalnie programowej obsługi, co przy częstych, krótkich i szybkich transmisjach powoduje znaczne obciążenie czasu pracy mikrokontrolera. Jednocześnie, przy większej liczbie komunikujących się ze sobą procesorów znaczna część czasu poświęconego na programową analizę odbieranych adresów jest tracona bezużytecznie, ponieważ tylko niewielki procent przesyłanej informacji jest przeznaczony dla danego mikrokontrolera. Automatyczne rozpoznawanie adresów umożliwia zmniejszenie rozmiaru oprogramowania obsługującego łącze szeregowo, a jednocześnie istotnie ogranicza liczbę nieefektywnych odwołań do procedur obsługujących port szeregowo.

Automatyczne rozpoznawanie adresów może być stosowane w 1, 2 i 3 trybie pracy łącza i jest uaktywniane przez ustawienie bitu SM2 w rejestrze SCON (patrz rozdz. 3.3.3). W trybie 2 i 3 pracy łącza rozpoznawaniu podlegają tylko te ramki, w których 9. bit danych jest jedynką. W trybie 1 pracy łącza są analizowane wszystkie odebrane bajty, które mają prawidłowy bit stopu. Przerwanie od odbiornika portu szeregowo jest aktywizowane tylko wtedy, gdy rozpoznawany (odebrany) bajt jest zgodny



Rozszerzone łącze szeregowo jest standardowym elementem w mikrokontrolerach C52 firmy Intel. Jednakże znaczna część innych producentów w produkowanych przez siebie mikrokontrolerach typu C52, a tym bardziej C51 umieszcza podstawowy wariant interfejsu szeregowo.

z zaprogramowaną maską adresu. Maską adresu jest określana stanem rejestrów SADDR i SADEN. Odebrany bajt jest zgodny z maską adresową wtedy, gdy:

$$(\text{bajt} \oplus \text{SADDR}) \cdot \text{SADEN} = 00\text{h} \quad (\text{operacja realizowana bitowo})$$

W praktyce oznacza to, że rejestr SADDR zawiera właściwą maskę adresową, a zera w rejestrze SADEN wskazują bity, których wartość w analizowanym bajcie nie ma znaczenia. Na przykład jeśli:

SADDR=1 1 0 0 0 0 1 oraz

SADEN=1 1 1 1 1 1 0, to

adres aktywny=1 1 0 0 0 0 X (gdzie X oznacza dowolną wartość bitu).

Takie rozwiązanie umożliwia (przy odpowiednim wyborze wartości wpisanych do rejestrów SADEN i SADDR) jednoczesne adresowanie kilku mikrokontrolerów, przy pominięciu pozostałych. Jeśli bowiem przyjmie się, że w przypadku drugiego mikrokontrolera będzie:

SADDR=1 1 0 0 0 1 0 oraz

SADEN=1 1 1 1 1 0 1, czyli

adres aktywny=1 1 0 0 0 0 X 0,

to łatwo można sprawdzić, że wysłanie adresu 11000001 spowoduje uaktywnienie tylko pierwszego mikrokontrolera, adres 11000010 uaktywni tylko drugi mikrokontroler, adres 11000000 odebrany zostanie przez oba mikrokontrolery, a każdy adres postaci XXXXXX11 zostanie przez oba mikrokontrolery zignorowany. Opisany przykład może być łatwo zaadaptowany do większej liczby komunikujących się ze sobą mikrokontrolerów.

3.3.5. Interfejs SPI

Interfejs SPI jest rodzajem interfejsu szeregowego, stosowanym m.in. w mikrokontrolerach S51, S52, S53 i S8252. Charakterystycznymi cechami interfejsu SPI są:

- przesyłanie danych wyłącznie w trybie synchronicznym,
- realizacja transmisji wykorzystująca 3 linie sygnałowe,
- możliwość jednoczesnego nadawania i odbioru danych (*full-duplex*),
- możliwość pracy w trybie urządzenia nadrzędnego lub podrzędnego,
- maksymalna prędkość transmisji równa 1/4 częstotliwości zegarowej mikrokontrolera (jednak nie więcej niż 1,5 Mb/s),
- możliwość sterowania prędkością transmisji (4 różne wartości),
- możliwość wyboru kolejności przesyłania bitów (od najstarszego do najmłodszego lub odwrotnie),
- możliwość wykorzystania interfejsu SPI do programowania pamięci programu i pamięci danych typu EEPROM mikrokontrolera.

Interfejs SPI wykorzystuje cztery wyprowadzenia mikrokontrolera: MOSI (P1.5), MISO (P1.6), SCK (P1.7) i $\overline{\text{SS}}$ (P1.4). Pierwsze trzy z wymienionych wyprowadzeń pełnią rolę linii sygnałowych, ostatnie zaś służy do aktywizacji interfejsu SPI w wybranym mikrokontrolerze pracującym jako urządzenie podrzędne. W przypadku interfejsu SPI, jako urządzenie nadrzędne rozumiany jest układ, który wytwarza sygnał



W niektórych układach (np. S51, S52), w których łączy SPI służy wyłącznie do programowania mikrokontrolera, wyprowadzenie \overline{SS} nie występuje.

taktujący transmisji synchronicznej, zaś urządzeniem podrzędnym jest układ, który wykorzystuje sygnał taktujący doprowadzony z zewnątrz.

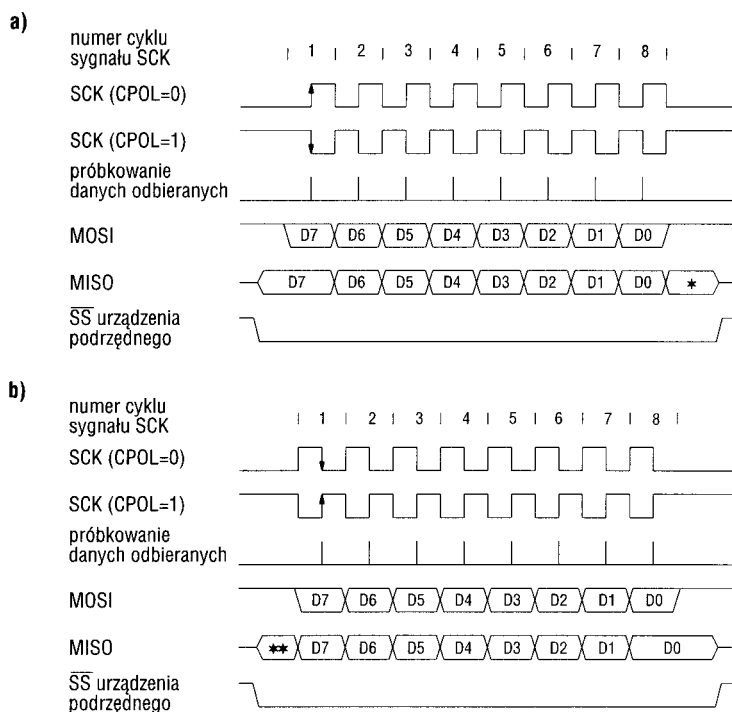
Możliwość programowego ustalania wielu parametrów pracy interfejsu SPI powoduje, że bardzo dobrze nadaje się on zarówno do współpracy ze specjalizowanymi układami peryferyjnymi o wejściu lub wyjściu szeregowym (mogą to być np. przetworniki A/C lub C/A), jak i do realizacji szybkiej komunikacji wieloprocessorowej (głównie w systemach zawierających pojedynczy procesor nadrzędny i jeden lub wiele procesorów podrzędnych). W przypadku wykorzystywania interfejsu SPI do współpracy ze specjalizowanymi układami peryferyjnymi, sposób jego podłączenia i sterowania zależy w znacznej mierze od rodzaju interfejsu i protokołu transmisji układu peryferyjnego. Przy określaniu sposobu podłączenia układów peryferyjnych do mikrokontrolera wyposażonego w interfejs SPI należy pamiętać, że przy pracy jako urządzenie nadrzędne wyprowadzenie SCK mikrokontrolera jest wyjściem sygnału taktującego, MOSI – wyjściem, a MISO – wejściem danych, a podczas pracy w trybie urządzenia podrzędnego SCK jest wejściem sygnału taktującego, MOSI – wejściem, a MISO – wyjściem danych. Uwzględniając te ustalenia łatwo dochodzi się do wniosku, że jeśli interfejs SPI ma być zastosowany do realizacji komunikacji wieloprocessorowej, to linie sygnałowe interfejsu należy wykonać jako połączenia wyprowadzeń MISO (jedna linia sygnałowa), MOSI (druga linia) i SCK (trzecia linia). Wyprowadzenie \overline{SS} mikrokontrolera funkcjonującego jako procesor nadrzędny nie jest wykorzystywane przez interfejs SPI i można je używać jak zwykłą linię wejść/wyjść. W pozostałych mikrokontrolerach wyprowadzenie \overline{SS} można traktować bądź jako wejście aktywizujące pracę interfejsu danego procesora podrzędnego (w przypadku wielu mikrokontrolerów pracujących w trybie urządzenia podrzędnego), bądź wymusić na nim na stałe stan niski (jeśli w systemie występuje tylko jeden mikrokontroler pracujący w trybie urządzenia podrzędnego). Wyprowadzenie \overline{SS} w mikrokontrolerach pracujących jako urządzenia podrzędne pełni zatem rolę zbliżoną do roli wyprowadzeń \overline{CE} (\overline{CS}) stosowanych w pamięciach RAM i EPROM, czy niektórych układach peryferyjnych. Przy wysokim stanie wyprowadzenia \overline{SS} interfejs SPI jest więc nieaktywny i w takim przypadku mikrokontrolery pracujące jako urządzenia podrzędne mogą wykorzystywać linię MOSI jako zwykłe wejście cyfrowe.

Sposób działania interfejsu zależy oczywiście od tego, czy interfejs pracuje w trybie urządzenia nadrzędnego, czy podrzędnego. W pierwszym przypadku (praca jako urządzenie nadrzędne) zapis do rejestru SPDR (jest to rejestr danych interfejsu SPI, wykonany jako 8-bitowy rejestr przesuwany) rozpoczyna transmisję danych. Kolejne bity bajtu danych umieszczonego w rejestrze SPDR są wysuwane i wysyłane linią MOSI, synchronicznie z generowanym na wyprowadzeniu SCK sygnałem taktującym. W tym samym czasie na wejście rejestru przesuwanego jest dołączone wyprowadzenie MISO mikrokontrolera, tak więc jednocześnie z wysunięciem każdego bitu nadawanych danych wczytywany (wsuwany) jest kolejny bit odbieranych danych. Ośmiokrotne powtórzenie operacji przesuwu powoduje zatem z jednej strony wysłanie do urządzenia podrzędnego bajtu wpisanego uprzednio do rejestru SPDR, z drugiej zaś strony umieszczenie w rejestrze SPDR bajtu odebranego od urządzenia podrzędnego. Po wykonaniu ośmiu przesunięć

generator sygnału taktującego (wysyłanego linią SCK) jest zatrzymywany i ustawiany jest bit SPIF (w rejestrze SPSR), będący wskaźnikiem przerwania od interfejsu SPI. Warunkiem zgłoszenia przerwania jest ustawienie bitu zezwolenia na przerwanie od interfejsu SPI (bit SPIE w rejestrze SPCR) i bitu zezwolenia na przerwanie od układu łącza szeregowego (bit ES w rejestrze IE).

Jeśli interfejs pracuje w trybie urządzenia podrzędnego, to cała operacja nadawania i odbierania danych wygląda podobnie z tą jednak różnicą, że taktowanie odbywa się z wykorzystaniem sygnału zewnętrznego (wytwarzanego przez układ pracujący w trybie urządzenia nadrzędnego). Dlatego też podczas pracy w trybie urządzenia podrzędnego interfejs nie może samorzutnie decydować o momencie rozpoczęcia transmisji – może on jedynie wpisać do rejestru SPDR bajt przeznaczony do wysłania, poczekać cierpliwie na transmisję taktowaną przez urządzenie nadrzędne i następnie odczytać z rejestru SPDR odebrany bajt danych.

O tym, czy interfejs pracuje w trybie urządzenia nadrzędnego, czy podrzędnego decyduje stan bitu MSTR, umieszczonego w rejestrze SPCR. Rejestr SPCR zawiera też kilka innych bitów definiujących sposób pracy interfejsu SPI. Należą do nich bity SPR0 i SPR1 określające prędkość transmisji, bit DORD, od którego zależy kolejność przesyłanych bitów (od najstarszego do najmłodszego lub odwrotnie) oraz bity CPOL



Rys. 3.13. Zależności czasowe występujące podczas transmisji danych pomiędzy dwoma mikrokontrolerami wykorzystującymi interfejs SPI: a) dla CPHA=0 i DORD=0; b) dla CPHA=1 i DORD=0

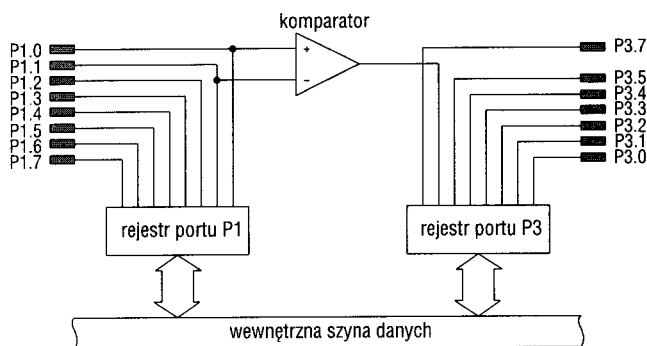
* wartość nieokreślona, zwykle bit D7 ostatnio odebranego bajtu danych

** wartość nieokreślona, zwykle bit D0 ostatnio odebranego bajtu danych

i CPHA określające polaryzację i aktywne zbocze sygnału taktującego (rys. 3.13). W rejestrze SPCR znajduje się również bit SPE, którego stan decyduje o tym, czy funkcja interfejsu SPI jest w ogóle aktywizowana.

3.4. Komparatory analogowe

W systemach mikroprocesorowych, zwłaszcza pełniących funkcję sterowników przemysłowych, zachodzi często potrzeba zastosowania komparatorów analogowych. Wśród układów rodziny '51 są takie, które mają komparatory analogowe zintegrowane ze strukturą mikrokontrolera. Bardzo często komparatory wykorzystywane są do monitorowania napięcia zasilającego – tego rodzaju elementy omówiono w rozdz. 3.6.2. Przykładem mikrokontrolerów zawierające klasyczne komparatory analogowe ogólnego zastosowania mogą być układy C1051, C2051, C4051.



Rys. 3.14. Wewnętrzne połączenia komparatora analogowego występującego w układach C1051, C2051 i C4051

Mikrokontrolery C1051, C2051 i C4051 zawierają pojedynczy komparator analogowy (rys. 3.14), którego wejścia podłączone są do linii P1.0 (+) i P1.1 (-). Z tego względu linie te pozbawione są standardowych rezystorów podciągających, a ich wejściowy prąd upływu nie przekracza $10\mu\text{A}$. Napięcie niezrównoważenia komparatora jest mniejsze od 20 mV, a zakres jego napięć wejściowych jest ograniczony od dołu potencjałem masy, a od góry napięciem zasilania mikrokontrolera. Stan wyjścia komparatora może być testowany wyłącznie programowo, przy czym operacja ta jest realizowana dość nietypowo – przez odczyt bitu P3.6 (port P3 nie ma wyprowadzenia P3.6). Bit P3.6 ma wartość jedynki logicznej, jeśli napięcie na wejściu nieodwracającym jest wyższe od napięcia na wejściu odwracającym.

3.5. Pamięć danych typu EEPROM

Niektóre spośród obecnie produkowanych mikrokontrolerów rodziny '51 mają wewnętrzną pamięć danych typu EEPROM. Pamięć danych typu EEPROM jest przydatna przede wszystkim wtedy, gdy sposób funkcjonowania konstruowanego urządzenia zależy od pewnych parametrów, których wartości zmieniane są niezbyt często. Przykładem takich parametrów mogą być nastawy początkowe urządzenia zmieniane przez użytkownika stosunkowo rzadko (np. pamięć częstotliwości odbieranych stacji w radiodbiorniku lub odbiorniku telewizyjnym) lub współczynniki urządzenia pomiarowego

zmieniane jedynie podczas okresowej kalibracji przyrządu. Mikrokontrolerami wyposażonymi w pamięć danych typu EEPROM są m.in. układy C851 i S8252.

3.5.1. Pamięć danych typu EEPROM w mikrokontrolerze S8252

Pamięć danych typu EEPROM w mikrokontrolerze S8252 ma pojemność 2 KB, a wartość każdego bajtu tej pamięci może być zmodyfikowana co najmniej 100 000 razy. Liczba cykli odczytu zawartości pamięci EEPROM jest nieograniczona.

Sterowanie pamięcią danych typu EEPROM w mikrokontrolerze S8252 jest niezwykle proste, ponieważ dostęp do tej pamięci jest uzyskiwany za pomocą rozkazów typu MOVX wykorzystujących wskaźnik DPTR. Rozróżnienie między instrukcją odwołującą się do obszaru zewnętrznej pamięci danych, a operacją wykonywaną na wewnętrznej pamięci danych typu EEPROM odbywa się na podstawie stanu bitu EEMEN umieszczonego w rejestrze WMCON. Jeśli wymieniony bit jest ustawiony, to operacja dotyczy pamięci EEPROM, w przeciwnym razie wykonywana jest operacja odwołująca się do zewnętrznej pamięci danych. Jeśli wykonywaną operacją ma być zapis do pamięci EEPROM, to oprócz ustawienia bitu EEMEN konieczne jest również ustawienie bitu EEMWE (także w rejestrze WMCON). Bit ten powinien być programowo wyzerowany po zakończeniu operacji zapisu do pamięci EEPROM. Obszar adresowy zajmowany przez pamięć EEPROM rozciąga się od adresu 0000h do 07FFh.

Charakterystyczną cechą pamięci typu EEPROM jest to, że odczyt takiej pamięci jest szybki i może być wykonywany z wykorzystaniem zwykłej instrukcji odczytu danych, ale czas trwania zapisu to zwykle milisekundy. Tak samo jest w przypadku pamięci danych typu EEPROM występującej w mikrokontrolerze S8252 – operacja zapisu jest w niej kontynuowana jeszcze przez około 2,5 ms od chwili wystąpienia rozkazu typu MOVX @DPTR, A (wykonanej przy ustawionych bitach EEMEN i EEMWE). Tak więc rozkaz MOVX @DPTR, A służy jedynie do wyzwalania operacji zapisu. Koniec rzeczywistej operacji zapisu do pamięci EEPROM można wyznaczyć testując stan bitu RDY/BSY umieszczonego w rejestrze WMCON. Bit ten jest sprzętowo zerowany na czas trwania operacji zapisu do pamięci EEPROM i sprzętowo ustawiany w momencie jej zakończenia. Inna metoda wyznaczania końca operacji zapisu bajtu danych do pamięci EEPROM polega na wykonywaniu prób odczytu bajtu pamięci EEPROM spod adresu, pod który następuje właśnie zapis. Podczas trwania zapisu najstarszy bit odczytywanego bajtu będzie różnił się od wartości wpisanej za pomocą rozkazu MOVX @DPTR, A, zaś identyczna wartość najstarszego bitu odczytanego i zapisanego bajtu jest jednoznacznym wskaźnikiem zakończenia wewnętrznej operacji zapisu do pamięci EEPROM. Wyposażenie pamięci EEPROM w mechanizmy umożliwiające określenie chwili zakończenia zapisu było konieczne, ponieważ wewnętrzny układ sterowania zapisem do pamięci EEPROM może w danej chwili obsługiwać tylko jedną operację zapisu, tak więc rozpoczęcie kolejnej operacji zapisu do tej pamięci nie może nastąpić przed zakończeniem poprzedniej.

Pamięć danych typu EEPROM występująca w mikrokontrolerach S8252 może być programowana nie tylko przy użyciu rozkazu typu MOVX @DPTR, A, ale także za pomocą zewnętrznych programatorów lub w specjalnym trybie programowania z wykorzystaniem synchronicznego łącza szeregowego (rozdz. 6.2.4).

3.6. Układy czuwające

Stosowanie układów czuwających (*watchdog*) ma zazwyczaj na celu podwyższenie niezawodności projektowanego systemu mikroprocesorowego. Z tego względu układy czuwające wykorzystuje się przede wszystkim wtedy, gdy mikrokontroler ma pracować w środowisku o znacznym poziomie zakłóceń. W obecnie produkowanych mikrokontrolerach spotyka się dwa podstawowe rodzaje układów czuwających – liczniki czuwające (*watchdog timer*) i generatory czuwające (*oscillator watchdog*). Do układów czuwających można zaliczyć również układy monitorowania napięcia zasilającego.

3.6.1. Liczniki czuwające

Liczniki czuwające są realizowane jako układy zliczające ilość cykli maszynowych. Przepełnienie takiego licznika powoduje wyzerowanie mikrokontrolera. Oprogramowanie mikrokontrolera wykorzystującego układ licznika czuwającego musi być zatem skonstruowane tak, by licznik czuwający był co pewien czas (krótszy niż czas po jakim występuje przepełnienie licznika) zerowany (odświeżany), gdyż tylko w taki sposób można zapobiec wyzerowaniu mikrokontrolera. Jeśli natomiast z jakiegoś powodu, np. wskutek wystąpienia silnych zakłóceń elektrycznych, mikrokontroler przestanie funkcjonować prawidłowo (wystąpi np. zjawisko analogiczne do tzw. zawieszenia się programu, znanego doskonale znacznej części użytkowników komputerów), licznik czuwający nie zostanie wyzerowany (odświeżony) w odpowiednim czasie i jego przepełnienie spowoduje wyzerowanie mikrokontrolera, a tym samym rozpoczęcie wykonywania programu przy standardowych warunkach początkowych. W niektórych licznikach czuwających operację zerowania zastąpiono przeładowywaniem licznika określoną wartością. Zwiększanie tej wartości powoduje oczywiście skracanie czasu jaki pozostał do chwili przepełnienia licznika. Tym sposobem uzyskuje się możliwość programowania maksymalnego czasu odświeżania zawartości licznika czuwającego. Oczywiście licznik może być przeładowywany wartością równą zeru, przy której maksymalny czas odświeżania jest najdłuższy. Zerowanie jest więc szczególnym przypadkiem odświeżania licznika czuwającego.

W mikrokontrolerach S53 i S8252 licznik czuwający taktowany jest za pomocą niezależnego, wewnętrznego generatora, współpracującego z preskalerem umożliwiającym zmianę maksymalnego czasu odświeżania w zakresie od 16 ms do 2048 ms. Stopień podziału preskalera zależy od stanu bitów PS0...PS2 znajdujących się w rejestrze WCON (WMCON), a stałość częstotliwości generatora taktującego licznik czuwający jest rzędu $\pm 30\%$ (przy napięciu zasilania równym 5,0 V). Licznik jest uruchamiany przez ustawienie bitu WDTEN umieszczonego w rejestrze WCON i może być zatrzymywany przez wyzerowanie bitu WDTEN. Licznik jest zatrzymywany na czas zamrożenia mikrokontrolera. Odświeżanie (zerowanie) omawianego licznika czuwającego następuje w wyniku ustawienia bitu WDTRST (rejestr WCON) i jest wykonywane z opóźnieniem jednego cyklu maszynowego w stosunku do momentu wyzerowania bitu WDTRST.



Operacje odświeżania licznika czuwającego nie powinny być wykonywane w ramach procedur obsługi przerw, ponieważ przerwy mogą funkcjonować prawidłowo, pomimo błędnego działania programu głównego.

Nieco inaczej wygląda sterowanie licznikiem czuwającym w układach S51, S52. Po wyzerowaniu mikrokontrolera licznik ten jest zablokowany. W celu jego uruchomienia należy wpisać do rejestru WDTRST sekwencję dwóch kodów (1Eh i 0E1h), która powoduje rozpoczęcie zliczania. Licznik czuwający w mikrokontrolerach S51 i S52 jest 14-bitowym licznikiem zliczającym (w górę) cykle maszynowe, tak więc jego maksymalny czas odświeżania zależy od częstotliwości taktowania mikrokontrolera. W celu odświeżenia licznika należy wykonać zapis do rejestru WDTRST sekwencji takiej samej, jak użyta do uruchomienia licznika. Rejestr WDTRST jest rejestrem przeznaczonym wyłącznie do zapisu, a bieżącego stanu licznika czuwającego nie można określić, ani zmienić przez odczyt lub zapis tego (bądź innego) rejestru. W przypadku przepełnienia licznika następuje wyzerowanie mikrokontrolera, przy czym (jeśli bit DISRTO w rejestrze AUXR jest zerem) jednocześnie generowany jest, trwający 98 cykli zegarowych, impuls zerujący na wyprowadzeniu RST. Impuls ten może być wykorzystany np. do zerowania zewnętrznych układów peryferyjnych. Podczas zamrożenia mikrokontrolera licznik czuwający nie zlicza (ze względu na zatrzymanie taktowania mikrokontrolera). Należy jednak zwrócić uwagę na fakt, że w układach S51 i S52 wyjście ze stanu zamrożenia może być realizowane także za pomocą przerw zewnętrznych wyzwalanych poziomem. Ponieważ w takim przypadku aktywny stan przerwania zewnętrznego musi być utrzymywany stosunkowo długo (co najmniej do czasu uruchomienia i ustabilizowania się generatora sygnału zegarowego), przez cały czas trwania niskiego (aktywnego) stanu przerwania wykorzystanego do wyprowadzenia mikrokontrolera ze stanu zamrożenia licznik czuwający nie jest inkrementowany. Pracą licznika czuwającego (tj. jego zliczaniem lub wyłączeniem) w trybie uśpienia steruje bit WDIDLE (rejestr AUXR).

3.6.2. Układy monitorowania napięcia zasilającego

Układy monitorowania napięcia zasilającego służą do sygnalizowania spadku napięcia zasilania mikrokontrolera poniżej pewnej wartości granicznej.

Rejestr PCON niektórych mikrokontrolerów zawiera specjalny bit POF, który jest ustawiany sprzętowo, jeśli wystąpiło załączenie napięcia zasilania lub chwilowy zanik tego napięcia (np. spadek poniżej wartości 3,0 V dla mikrokontrolerów zasilanych napięciem 5,0 V). Testowanie stanu bitu POF i wskaźnika przepełnienia licznika czuwającego pozwala zatem na określenie przyczyny wyzerowania mikrokontrolera. Stan bitu POF nie ulega zmianie w wyniku zerowania mikrokontrolera, jeśli nie wystąpił w tym czasie spadek napięcia zasilającego poniżej danej wartości progowej (czyli jeśli zerowanie jest skutkiem np. przepełnienia licznika czuwającego).



Bit POF w rejestrze PCON nie występuje we wszystkich mikrokontrolerach o danym oznaczeniu – o implementacji tego bitu decyduje producent układu. Bit POF stanowi np. standardowe wyposażenie mikrokontrolerów C52 firmy Intel i najnowszych wersji mikrokontrolerów C51 i C52 firmy Philips, ale układy starsze lub układy innych producentów o analogicznych oznaczeniach mogą nie realizować opisanej funkcji.

Układ przerw



Idea przerwai w mikrokontrolerach polega na tym, że w odpowiedzi na określony sygnał (sygnał przerwania) mikrokontroler zawiesza chwilowo wykonywanie programu głównego i wykonuje specjalną procedurę określaną mianem procedury obsługi przerwania. Po zakończeniu tej procedury mikrokontroler wraca do wykonywania programu głównego, począwszy od miejsca, w którym zostało ono zawieszone.

Podstawową funkcją przerwai jest umożliwienie szybkiego reagowania na zdarzenia zewnętrzne. Zaletą wykorzystywania przerwai jest też uproszczenie oprogramowania. Wynika ono m.in. stąd, że program główny jest uwalniany od konieczności testowania stanu wybranych układów peryferyjnych lub linii sygnałowych, ponieważ w razie stosowania przerwai operacje te są realizowane sprzętowo.

W niektórych mikroprocesorach jedynym rodzajem przerwai są przerwania zewnętrzne. W mikrokontrolerach, oprócz przerwai zewnętrznych, występują też z reguły przerwania od wewnętrznych układów peryferyjnych. Mikrokontrolery rodziny '51 nie stanowią pod tym względem wyjątku.

4.1. Struktura układu przerwai

W mikrokontrolerach rodziny '51 sygnałem powodującym przerwanie jest ustawienie odpowiedniego wskaźnika przerwania, czyli bitu zgłaszającego przerwanie. W mikrokontrolerach tych każde źródło przerwania ma swój własny wskaźnik przerwania (tab. 4.1). Liczba źródeł przerwai, a tym samym wskaźników przerwai, zależy więc od typu mikrokontrolera, a konkretnie od ilości i możliwości wewnętrznych układów peryferyjnych danego mikrokontrolera. Wskaźniki przerwai umieszczone są w rejestrach SFR.

Wskaźniki przerwai ustawiane są sprzętowo, wskutek wystąpienia określonych zdarzeń – np. przepełnienia licznika lub odebrania bajtu informacji przez łącze szeregowe. Niektóre ze wskaźników mogą być także ustawiane programowo – właściwość tę można wykorzystać do programowego symulowania pewnych zdarzeń lub do realizacji przerwai programowych (nie występują one na liście instrukcji mikrokontrolerów rodziny '51). Każdy wskaźnik przerwania powinien być zerowany (sprzętowo lub programowo) przed zakończeniem procedury obsługi danego przerwania, ponieważ

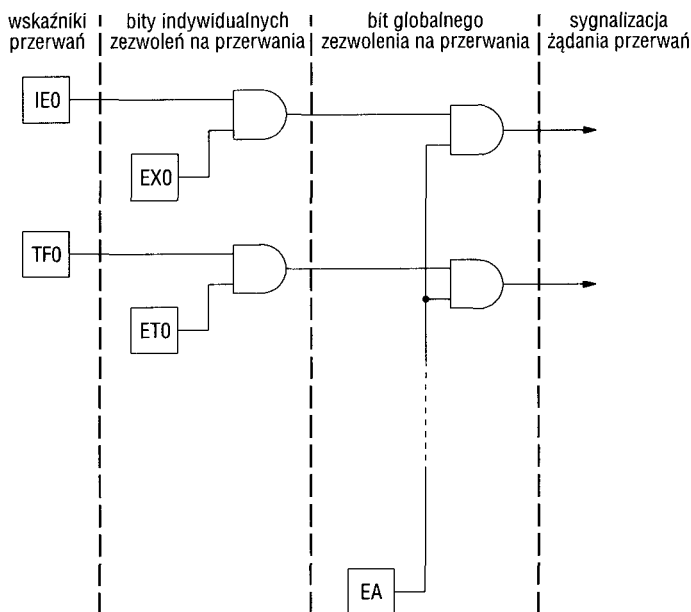
Tab. 4.1. Zestawienie przerwai występujących w wybranych mikrokontrolerach rodziny '51

Źródło przerwania	Wskaźniki przerwania	Wektor przerwania	Uwagi
Przerwanie zewnętrzne INT0	IE0	0003h	
Przerwanie zewnętrzne INT1	IE1	0013h	
Układ licznikowy T0	TF0	000Bh	
Układ licznikowy T1	TF1	001Bh	z wyjątkiem C1051
Układ licznikowy T2	TF2+EXF2	002Bh	w mikrokontrolerach wyposażonych w licznik T2
Interfejs szeregowy	RI+TI	0023h	z wyjątkiem C1051
Interfejs szeregowy i interfejs SPI	RI+TI+SPIF	0023H	S53, S8252

w przypadku pozostawienia wskaźnika ustawionego, procedura obsługi przerwania zostanie wywołana ponownie (układ przerwań nie reaguje na moment ustawiania wskaźnika, lecz na jego stan). Wyzerowanie wskaźnika przerwania jest w przypadku niektórych przerwań realizowane sprzętowo (automatycznie) – w chwili przejścia do procedury obsługi przerwania. Część wskaźników przerwań wymaga jednak zerowania programowego.

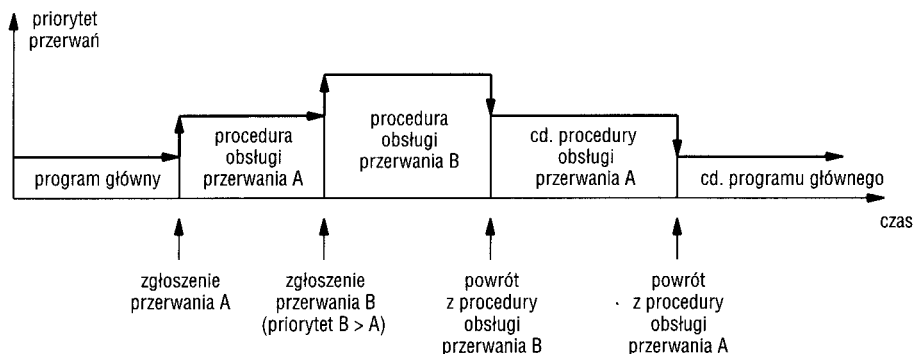
Ustawienie wskaźnika przerwania jest warunkiem koniecznym, ale nie wystarczającym do zgłoszenia przerwania. Każdemu wskaźnikowi przerwania towarzyszy bowiem bit zezwolenia na przerwanie i żeby możliwe było zgłoszenie danego przerwania, bit ten musi być ustawiony. Jeśli bit zezwolenia na przerwanie będzie zerem, wszystkie żądania przerwań pochodzące z danego źródła przerwań będą blokowane (ignorowane). W mikrokontrolerach rodziny '51, oprócz indywidualnych bitów zezwolenia na przerwanie, występuje dodatkowo bit globalnego zezwolenia na przerwanie. Bit ten jest oznaczony symbolem EA i, podobnie jak indywidualne bity zezwoleń na przerwanie, jest on umieszczony w obszarze rejestrów SFR (jest to najbardziej znaczący bit rejestru IE). Wyzerowanie bitu EA powoduje zablokowanie wszystkich przerwań. Żeby umożliwić obsługę wybranych przerwań, należy zatem ustawić odpowiednie bity indywidualnych zezwoleń na przerwanie oraz bit EA globalnego zezwolenia na przerwanie (rys. 4.1). Standardowo po wyzerowaniu mikrokontrolera, wszystkie bity zezwoleń na przerwanie (zarówno indywidualne, jak i bit globalnego zezwolenia na przerwanie) są wyzerowane.

Każdy z mikrokontrolerów rodziny '51 ma co najmniej kilka źródeł przerwań. W przypadku jednoczesnego zgłoszenia dwóch lub więcej przerwań układ przerwań



Rys. 4.1. Fragment struktury układu przerwań przedstawiający sposób funkcjonowania wskaźników przerwań i bitów zezwolenia na przerwanie

musi jednoznacznie określić, które ze zgłoszonych przerwań ma być obsługiwane jako pierwsze – decyduje o tym priorytet przerwań. W pierwszej kolejności rozpoczynana jest obsługa przerwania o najwyższym priorytecie, zaś obsługa przerwań o niższym priorytecie rozpocznie się dopiero po zakończeniu obsługi przerwań o wyższym priorytecie. Jeśli podczas wykonywania procedury obsługi przerwania zostanie zgłoszone przerwanie o priorytecie wyższym od aktualnie obsługiwanego, to wykonywanie bieżącej procedury obsługi przerwania zostanie zawieszona i mikrokontroler rozpocznie obsługę nowego przerwania (o wyższym priorytecie). Dopiero po zakończeniu tej ostatniej nastąpi powrót do zawieszona obsługi przerwania o niższym priorytecie (rys. 4.2).



Rys. 4.2. Przykład wykorzystania czasu pracy CPU przez program główny i procedury obsługi przerwań

Efektywny priorytet danego przerwania, decydujący o kolejności obsługi przerwań, zależy od dwóch czynników:

- poziomu priorytetu danego przerwania,
- naturalnego priorytetu przerwań.

Podczas określania efektywnego priorytetu przerwania, pierwszy z tych czynników (poziomy priorytet) jest czynnikiem nadrzędnym w stosunku do drugiego (naturalnego priorytetu). Tak więc efektywny priorytet dowolnego przerwania o poziomie wyższym, będzie wyższy od priorytetu każdego z przerwań przypisanych do poziomu niższego. Oznacza to, że mikrokontroler nie rozpocznie obsługi żadnego z przerwań o poziomie niższym, jeśli jest zgłoszone choćby jedno przerwanie o poziomie wyższym.

Dla danego mikrokontrolera liczba poziomów priorytetu jest stała, zależy od typu mikrokontrolera i mieści się w zakresie od 1 do 4 (standardowo mikrokontrolery C51, C52, S51, S52, S53, S8252, C1051, C2051, C4051 mają dwa poziomy priorytetu, ale najnowsze wersje mikrokontrolerów C51 i C52 firmy Philips są wyposażone aż w cztery poziomy priorytetu). Poziom priorytetu poszczególnych przerwań zależy od stanu tzw. bitów priorytetu i może być zmieniany programowo. Bity określające poziom priorytetu poszczególnych przerwań najczęściej spotykanych mikrokontrolerów umieszczone są w rejestrze IP. W przypadku większej liczby źródeł przerwań lub poziomów priorytetu służy do tego celu więcej niż jeden rejestr. Oznaczenia dodatkowych rejestrów poziomów priorytetu mają zwykle dodany indeks (np. IP1) lub oznaczenie IPH.

Tab. 4.2. Naturalny priorytet przerw wybranych mikrokontrolerów rodziny '51

Źródło lub wskaźnik przerwania	C51 S51 LV51 C1051U C2051 C4051	C52 S52 LV52 S8252	C1051
INT0	1	1	1
INT1	3	3	3
Licznik T0	2	2	2
Licznik T1	4	4	
Licznik T2	6*	6	
łącze szeregowe	5	5	

* Jak wspomniano wcześniej, najnowsze wersje układów C51 produkowane przez firmę Philips są wyposażone w licznik T2 i od układów C52 różnią się wyłącznie zmniejszonym rozmiarem wewnętrznej pamięci programu i danych.

Naturalny priorytet przerw (tab. 4.2) jest to priorytet jaki przerwanie uzyskuje bezpośrednio po wyzerowaniu mikrokontrolera, czyli wtedy, gdy wszystkie bity określające poziom priorytetu poszczególnych przerw są w jednakowym stanie (są wyzerowane). Naturalny priorytet przerw zależy od typu mikrokontrolera i nie może być w żaden sposób zmieniony. Możliwe jest występowanie kilku przerw o identycznym priorytecie naturalnym – np. przerwanie od nadajnika i odbiornika łączy szeregowego. Przerwania takie mają wspólny wektor adresowy (adres początku procedury obsługi przerwania), ponieważ w przeciwnym razie nie byłoby możliwe określenie jednoznacznej reakcji mikrokontrolera na jednoczesne zgłoszenie takich przerw. Wektory adresowe poszczególnych przerw zależą od typu mikrokontrolera (tab. 4.1) i nie mogą być modyfikowane w żaden sposób. W większości mikrokontrolerów mających więcej niż jeden poziom priorytetu przerw, wszystkie przerwy mają indywidualne bity priorytetu (oczywiście przerwaniom o tym samym wektorze adresowym odpowiada wspólny bit priorytetu). Dzięki temu każde przerwanie może być niezależnie od pozostałych przypisane do wybranego poziomu priorytetu.

4.2. Obsługa przerw

Stan wskaźników przerw jest próbkowany w stanie S5P2 każdego cyklu maszynowego. Interpretacja spróbkowanego stanu wskaźników jest dokonywana w ciągu następnego cyklu maszynowego. Jeśli zostanie stwierdzone, że jakikolwiek ze wskaźników przerw jest ustawiony, a przerwanie to jest odblokowane (tzn. ustawiony jest odpowiedni indywidualny i globalny bit zezwolenia na przerwanie), to układ przerw spowoduje wykonanie sprzętowo wygenerowanej instrukcji dalekiego wywołania procedury. Adres tej procedury będzie określony przez wektor adresowy zgłoszonego przerwania. Opisana operacja zostanie jednak wstrzymana, jeśli układ przerw stwierdzi, że zachodzi dowolny z następujących przypadków:



Mikrokontroler nie pamięta faktu zgłoszenia przerwania, lecz w każdym cyklu maszynowym powtarza od początku opisane operacje, czyli na nowo sprawdza stan wskaźników przerw. Jeśli zatem przed ustaniem czynników wymienionych w punktach a...c (blokujących rozpoczęcie wykonywania procedury obsługi przerwania) wskaźnik przerwania powróci do stanu nieaktywnego, to zgłoszenie przerwania nie zostanie w ogóle zauważone, a tym samym nie będzie obsługiwane.

- a) wykonywana jest procedura obsługi przerwania o priorytecie identycznym lub wyższym od priorytetu zgłoszonego właśnie przerwania,
- b) bieżący cykl maszynowy nie jest ostatnim cyklem wykonywanej instrukcji,
- c) wykonywana jest instrukcja RETI lub dowolna inna instrukcja modyfikująca zawartość któregośkolwiek z rejestrów zawierających bity priorytetu lub bity zezwolenia na przerwanie.

W tym ostatnim przypadku układ przerw zostanie zablokowany na okres bieżącej i następnej instrukcji – przez ten czas stan układu przerw zostanie zaktualizowany.

Przejsie do procedury obsługi przerwania następuje w wyniku wykonania przez mikrokontroler wspomnianej, sprzętowo generowanej instrukcji dalekiego wywołania procedury. Instrukcja ta powoduje umieszczenie na stosie zawartości licznika rozkazów, nie zapamiętuje jednak zawartości rejestru wskaźników stanu (PSW). Tak więc ewentualne operacje zapamiętania i odtworzenia zawartości rejestru PSW muszą być wykonane programowo. Powrót z procedury obsługi przerwania następuje w wyniku wykonania rozkazu RETI. Instrukcja ta powoduje umieszczenie w liczniku rozkazów adresu powrotu z procedury (zapamiętanego uprzednio na stosie) oraz przywraca układ przerw do stanu jaki występował przed rozpoczęciem obsługi ostatniego przerwania. Należy zwracać uwagę, by na końcu procedury obsługi przerwania nie umieścić rozkazu zwykłego powrotu z procedury (RET), ponieważ rozkaz ten, w odróżnieniu od rozkazu RETI, nie odtwarza stanu układu przerw. Tym samym układ przerw nie zostanie poinformowany o zakończeniu obsługi przerwania i wszystkie później zgłoszone przerwy (o priorytecie takim samym lub niższym od właśnie zakończonego) będą zablokowane.

4.3. Czas reakcji na przerwanie

Minimalny czas reakcji na przerwanie wynika z samego sposobu realizacji przejścia do procedury obsługi przerwania. Przed wykonaniem pierwszej instrukcji tej procedury musi bowiem upłynąć co najmniej jeden cykl maszynowy oczekiwania na interpretację stanu spróbowanych wskaźników przerw oraz dwa cykle maszynowe podczas których jest wykonywana sprzętowo generowana instrukcja wywołania procedury obsługi przerwania. Tak więc, nawet przy najkorzystniejszych uwarunkowaniach, czas reakcji na przerwanie jest nieco dłuższy niż trzy cykle maszynowe. Maksymalny czas reakcji na przerwanie zależy natomiast od możliwości wystąpienia wymienionych w rozdz. 4.2 czynników wstrzymujących przejście do procedury obsługi przerwania. W szczególności obsługa przerwania o niskim priorytecie może być znacznie

UWAGA

Omówiony czas reakcji na przerwanie jest czasem liczonym dla standardowych mikrokontrolerów rodziny '51 (o 12 cyklach zegarowych na cykl maszynowy) od momentu zgłoszenia przerwania, czyli od chwili próbkowania stanu wskaźników przerw. W rozważaniach tych nie jest natomiast uwzględniany czas, jaki upływa od zdarzenia wywołującego przerwanie (np. moment przepełnienia licznika), do chwili ustawienia wskaźnika, a tym bardziej do momentu zgłoszenia przerwania (czyli próbkowania stanu wskaźników przerw).

opóźniona, jeśli w tym samym czasie pojawią się przerwania o wyższym priorytecie. Jeśli jednak przyjmie się założenie, że przejście do procedury obsługi przerwania nie będzie blokowane przez obsługę przerw o wyższym priorytecie, to czas reakcji na zgłoszenie przerwania, liczony na najgorszy przypadek, zostanie ograniczony do niecałych 9 cykli maszynowych. Taka sytuacja może wystąpić, jeśli przerwanie zostanie zgłoszone podczas wykonywania rozkazu zmieniającego stan układu przerw (np. rozkazu RETI), po którym mikrokontroler musi wykonać jeszcze jeden rozkaz, zanim przejdzie do obsługi następnego przerwania. W stosunku do wartości minimalnej (ponad 3 cykle maszynowe), czas reakcji na przerwanie może wówczas ulec zwiększeniu o jeden cykl przeznaczony na dokończenie bieżącego rozkazu (RETI) i nawet cztery cykle następnego rozkazu (jeśli będzie to rozkaz mnożenia lub dzielenia). Suma wyniesie wtedy ponad 8 cykli maszynowych.

4.4. Przerwania zewnętrzne

4.4.1. Przerwania $\overline{\text{INT0}}$ i $\overline{\text{INT1}}$

Wszystkie mikrokontrolery rodziny '51 (nawet te o zredukowanej liczbie wyprowadzeń) mają co najmniej dwa wejścia przerw zewnętrznych, oznaczane jako $\overline{\text{INT0}}$ i $\overline{\text{INT1}}$. W zależności od stanu bitów IT0 i IT1 (umieszczonych w rejestrze TCON), przerwania zewnętrzne z wejść $\overline{\text{INT0}}$ i $\overline{\text{INT1}}$ aktywowane są niskim poziomem sygnału zewnętrznego lub opadającym zboczem tego sygnału. Detekcja zbocza sygnału przerwania odbywa się synchronicznie z pracą mikrokontrolera, na drodze testowania stanu wejść przerw zewnętrznych w kolejnych cyklach maszynowych. Wykrycie wysokiego stanu w jednym cyklu maszynowym i niskiego stanu w następnym cyklu maszynowym jest interpretowane jako opadające zbocze testowanego sygnału. Skutkiem wykrycia zbocza jest ustawienie wskaźnika przerwania (bity IE0 i IE1 w rejestrze TCON). Ustawiony w ten sposób wskaźnik przerwania zewnętrznego jest zerowany sprzętowo w momencie przejścia do procedury obsługi przerwania.

W przypadku przerw $\overline{\text{INT0}}$ i $\overline{\text{INT1}}$ wyzwalanych poziomem, stan niski na wejściu przerwania musi trwać do momentu rozpoczęcia procedury obsługi przerwania, po czym sygnał zewnętrzny powinien powrócić do stanu wysokiego. Jeśli powrót do stanu wysokiego nie nastąpi przed końcem procedury obsługi przerwania, to układ

przerwań może uznać że zgłoszone zostało nowe przerwanie i ponownie wywołać procedurę obsługi tego samego przerwania. W przypadku przerwań $\overline{\text{INT0}}$ i $\overline{\text{INT1}}$ aktywizowanych poziomem, wskaźniki IE0 i IE1 są sterowane bezpośrednio przez stan wejść przerwań zewnętrznych i nie będą reagować na próby programowego ustawiania. Jedyną metodą programowego wyzwolenia przerwań $\overline{\text{INT0}}$ lub $\overline{\text{INT1}}$ aktywowanego poziomem jest więc zapis zera do odpowiedniego bitu rejestru portu P3, a nie ustawianie wskaźnika przerwania.

Zmniejszanie poboru mocy mikrokontrolerów

5



Pobór mocy stanowił zawsze istotny czynnik ograniczający możliwość zastosowań układów cyfrowych. Nie inaczej jest w przypadku mikrokontrolerów rodziny '51, które początkowo były wykonywane wyłącznie w technologii NMOS i pobierały prąd o natężeniu ok. 100...250 mA. Pierwszą próbą poprawy tej sytuacji było opracowanie wersji mikrokontrolerów mających specjalne wyprowadzenie umożliwiające zasilanie całości lub części wewnętrznej pamięci RAM mikrokontrolera przy odłączonym napięciu zasilania od pozostałej części mikrokontrolera. Takie rozwiązanie pozwalało na czasowe wyłączanie zasilania mikrokontrolera, co było możliwe w takich zastosowaniach, w których interwencje mikrokontrolera były stosunkowo rzadkie i stanowiły odpowiedź na zdarzenia zewnętrzne. Do podtrzymania pamięci danych w tych mikrokontrolerach wystarczał prąd zasilania kilka...kilkanaście mA, co stanowiło wartość przynajmniej 10-krotnie mniejszą od wartości prądu zasilania całego mikrokontrolera.

Pobór mocy mikrokontrolerów został radykalnie zmniejszony przez wprowadzenie technologii CMOS. Przy maksymalnej częstotliwości taktowania jest on kilka, a czasem kilkunastokrotnie mniejszy od poboru mocy takich samych mikrokontrolerów wykonanych w technologii NMOS. Ponadto, o ile w technologii NMOS częstotliwość taktowania ma znikomy wpływ na pobór mocy, to w przypadku układów CMOS prąd zasilania wykazuje prawie proporcjonalną zależność od częstotliwości taktowania, co daje możliwość dalszej redukcji poboru mocy mikrokontrolera przez zmniejszanie częstotliwości pracy. Wymienione cechy przesądziły o wyborze technologii. Obecnie wszystkie produkowane mikrokontrolery rodziny '51, które były wykonywane w technologii NMOS mają swoje odpowiedniki funkcjonalne w postaci układów CMOS. Ceny tych układów w wersji CMOS są przy tym porównywalne lub niższe od mikrokontrolerów wersji NMOS. Ostatecznie zalety technologii CMOS spowodowały, że niemal wszystkie spośród kilkudziesięciu różnych typów mikrokontrolerów rodziny '51 oferowane są wyłącznie w postaci układów CMOS.

Silna zależność poboru mocy od częstotliwości pracy układów CMOS umożliwiła wprowadzenie kilku, niespotykanych w mikrokontrolerach NMOS, trybów pracy ze zmniejszonym poborem mocy. Jednocześnie, o ile w przypadku układów NMOS tylko niektórzy producenci oferują niektóre ze swoich mikrokontrolerów w wersjach umożliwiających podtrzymywanie zawartości wewnętrznej pamięci RAM przy wyłączonym zasilaniu mikrokontrolera, to w mikrokontrolerach wykonanych w technologii CMOS tryby pracy ze zmniejszonym poborem mocy są standardowo implementowane we wszystkich mikrokontrolerach. W obecnie produkowanych mikrokontrolerach CMOS rodziny '51 spotyka się następujące tryby pracy ze zmniejszonym poborem mocy:

- uśpienie (*Idle*),
- zamrożenie (*Power Down*),
- praca zwolniona (*Slow Down*),
- tryb zamrożenia aktywowany sprzętowo (*Hardware Power Down*).

Najistotniejsze są pierwsze dwa z wymienionych, gdyż występują one we wszystkich częściej spotykanych mikrokontrolerach rodziny '51.

5.1. Tryb uśpienia (*Idle*)

Redukcja poboru mocy w stanie uśpienia następuje na skutek zablokowania sygnału taktującego CPU. CPU jest w związku z tym zasilane wyłącznie prądem spoczynkowym. Uśpienie mikrokontrolera nie przerywa jednak pracy generatora sygnału zegarowego, układu przerwań i z reguły także wewnętrznych układów peryferyjnych.

Po wprowadzeniu mikrokontrolera w uśpienie dalsza redukcja poboru mocy mikrokontrolera możliwa jest przez zmniejszenie liczby działających układów peryferyjnych. Podczas uśpienia zawartość wewnętrznej pamięci RAM mikrokontrolera oraz rejestrów SFR jest podtrzymywana, przy czym zawartość części rejestrów może być zmieniana na skutek działania niektórych peryferii i układu przerwań.

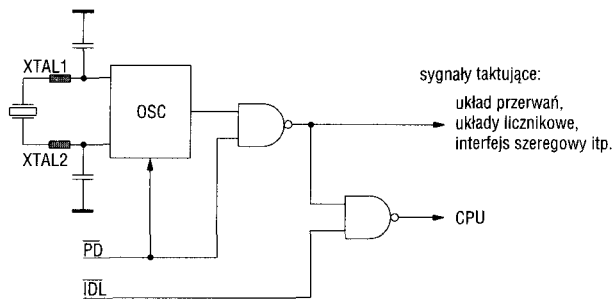
Po wejściu w uśpienie linie portów utrzymują stan jaki miały w chwili przejścia mikrokontrolera w uśpienie, jednakże wyprowadzenia pełniące funkcje alternatywne (typu wyjścia) aktywnych układów peryferyjnych kontynuują pracę. Dzięki temu uśpienie mikrokontrolera nie zakłóca działania wyjść związanych z układami licznikowymi, portów szeregowych itp. Wyjścia sygnałów ALE i PSEN są utrzymywane w stanie wysokim. Wszystkie wyprowadzenia mikrokontrolera wykorzystywane jako wejścia pracują bez zmian tak, że operacje przechwytywania, zliczania impulsów zewnętrznych, przerwania zewnętrzne itp. obsługiwane są tak samo, jak podczas aktywnej pracy mikrokontrolera.

Wprowadzenie mikrokontrolera w stan uśpienia następuje wskutek ustawienia bitu IDL w rejestrze PCON – instrukcja ustawiająca bit IDL jest przy tym ostatnią instrukcją wykonaną przed przejściem mikrokontrolera w stan uśpienia. W bardziej zaawansowanych mikrokontrolerach występują dodatkowe zabezpieczenia przed przypadkowym (np. wskutek zakłóceń) przejściem w tryby zmniejszonego poboru mocy (czyli m.in. w tryb uśpienia). Na przykład w niektórych układach Siemens'a wprowadzenie mikrokontrolera w stan uśpienia wymaga użycia podwójnej instrukcji. Pierwsza z tych instrukcji musi ustawiać bit IDLE, zapisując jednocześnie zero do bitu IDLS (oba bity znajdują się w rejestrze PCON). Druga instrukcja, następująca bezpośrednio po pierwszej, musi ustawiać bit IDLS, zerując jednocześnie bit IDLE – jest ona przy tym ostatnią instrukcją wykonywaną przed uśpieniem mikrokontrolera. Wszelkie inne próby wykorzystania bitów IDLE i IDLS, w szczególności zaś jednoczesne ustawienie obu bitów, nie spowoduje przejścia mikrokontrolera w tryb uśpienia, a bity IDLE i IDLS zostaną automatycznie wyzerowane.

Wyjście mikrokontrolera ze stanu uśpienia następuje bądź w wyniku sprzętowego wyzerowania procesora (sygnał zerowania mikrokontrolera musi trwać co najmniej dwa pełne cykle maszynowe) lub w wyniku przerwania (pod warunkiem, że bity zerzwolenia na przerwanie są ustawione). W tym ostatnim przypadku mikrokontroler wykonuje procedurę obsługi przerwania, a po wykonaniu instrukcji RETI przechodzi do wykonywania instrukcji znajdującej się bezpośrednio za instrukcją, która spowodowała uśpienie mikrokontrolera.

5.2. Tryb zamrożenia (Power Down)

Skutkiem przejścia mikrokontrolera w tryb zamrożenia jest zatrzymanie generatora sygnału zegarowego mikrokontrolera (rys. 5.1) – w związku z tym wszystkie operacje realizowane przez mikrokontroler i jego wewnętrzne układy peryferyjne są zatrzymywane. Zawartość wewnętrznej pamięci RAM i rejestrów SFR jest podtrzymywana. Linie portów mikrokontrolera pełniące funkcje alternatywne utrzymują stan, w którym znalazły się pod koniec ostatniego cyklu zegarowego instrukcji powodującej zamrożenie mikrokontrolera. Pozostałe linie portów utrzymują stan wynikający z wartości rejestrów portów. Wyprowadzenia ALE i PSEN utrzymywane są w stanie niskim (tab. 5.1).



Rys. 5.1. Struktura generatora sygnału zegarowego w mikrokontrolerze uwzględniająca zmiany w sposobie taktowania układów wewnętrznych mikrokontrolera po przejściu w stan uśpienia lub zamrożenia

Wprowadzenie mikrokontrolera w stan zamrożenia następuje w wyniku ustawienia bitu PD w rejestrze PCON. W przypadku niektórych nowszych opracowań może być konieczne, podobnie jak przy przechodzeniu w uśpienie, użycie sekwencji dwu instrukcji, ustawiających najpierw bit pomocniczy, a potem właściwy bit powodujący przejście w tryb zamrożenia. Próba jednoczesnego wymuszenia uśpienia i zamrożenia mikrokontrolera powoduje przejście w stan zamrożenia.

Podczas zamrożenia napięcie zasilania mikrokontrolera może być obniżone w celu dalszego zmniejszenia poboru mocy. Napięcie zasilania nie może być jednak obniżone przed wejściem mikrokontrolera w stan zamrożenia, musi też być przywrócone do normalnego poziomu przed wystąpieniem sygnału powodującego powrót mikrokontrolera do pracy aktywnej (przed wyjściem ze stanu zamrożenia). Minimalna wartość napięcia zasilania w stanie zamrożenia wynosi zwykle 2,0 V, ale w przypadku mikrokontrolerów niskonapięciowych może to być nawet 1,0 V.

Standardowo wyjście ze stanu zamrożenia następuje wskutek pojawienia się sygnału zerowania mikrokontrolera, który odblokowuje generator sygnału zegarowego. Sygnał

Tabela 5.1. Stan wyprowadzeń mikrokontrolerów rodziny '51 po przejściu w stan uśpienia lub zamrożenia, w zależności od położenia instrukcji powodującej przejście mikrokontrolera w tryb zmniejszonego poboru mocy

	Wewnętrzna pamięć programu		Zewnętrzna pamięć programu	
	uśpienie	zamrożenie	uśpienie	zamrożenie
ALE	wysoki	niski	wysoki	niski
PSEN	wysoki	niski	wysoki	niski
Linie portu P0	dane	dane	3. stan	3. stan
Linie portu P1	dane/funkcja alternatywna	dane/ostatnia wartość	dane/funkcja alternatywna	dane/ostatnia wartość
Linie portu P2	dane	dane	adres	dane
Linie portu P3	dane/funkcja alternatywna	dane/ostatnia wartość	dane/funkcja alternatywna	dane/ostatnia wartość
Linie innych portów	dane/funkcja alternatywna	dane/ostatnia wartość	dane/funkcja alternatywna	dane/ostatnia wartość

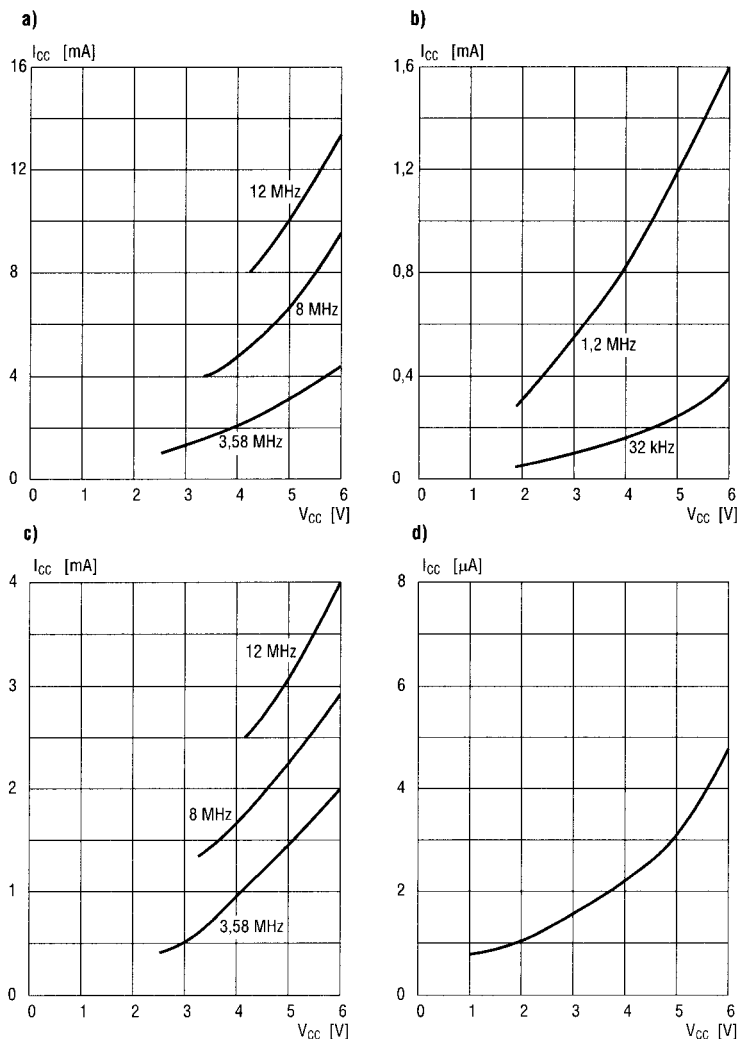
zerowania musi zatem trwać wystarczająco długo (podobnie jak przy włączaniu zasilania), aby możliwy był start i stabilizacja sygnału generatora taktującego. W przypadku stosowania generatora zewnętrznego, działającego przez cały czas (tj. także podczas zamrożenia mikrokontrolera), czas trwania sygnału zerującego może być krótszy, musi jednak umożliwiać przeprowadzenie pełnej procedury zerowania mikrokontrolera (tzn. trwać co najmniej dwa pełne cykle maszynowe). Zerowanie mikrokontrolera podczas wychodzenia ze stanu zamrożenia powoduje oczywiście zmianę zawartości znacznej części rejestrów SFR oraz restart programu, co niekiedy stanowi istotną wadę trybu zamrożenia. Niektóre mikrokontrolery umożliwiają jednak wyjście ze stanu zamrożenia nie tylko wskutek pojawienia się sygnału zerującego, lecz także w wyniku wystąpienia przerwania.

Niektóre mikrokontrolery (np. S51, czy najnowsze wersje C51, C52 firmy Philips) mogą być wyprowadzane ze stanu zamrożenia przez przerwania zewnętrzne INT0 lub INT1. W tym celu przerwania te należy odblokować i zaprogramować jako przerwania aktywowane poziomem (nie zboczem). Czas trwania stanu niskiego na wejściu takiego przerwania musi zapewniać restart i stabilizację pracy generatora zegarowego. Powrót mikrokontrolera do wykonywania programu nastąpi po powrocie wejścia przerwy do stanu wysokiego.

5.3. Inne możliwości zmniejszania poboru mocy mikrokontrolerów

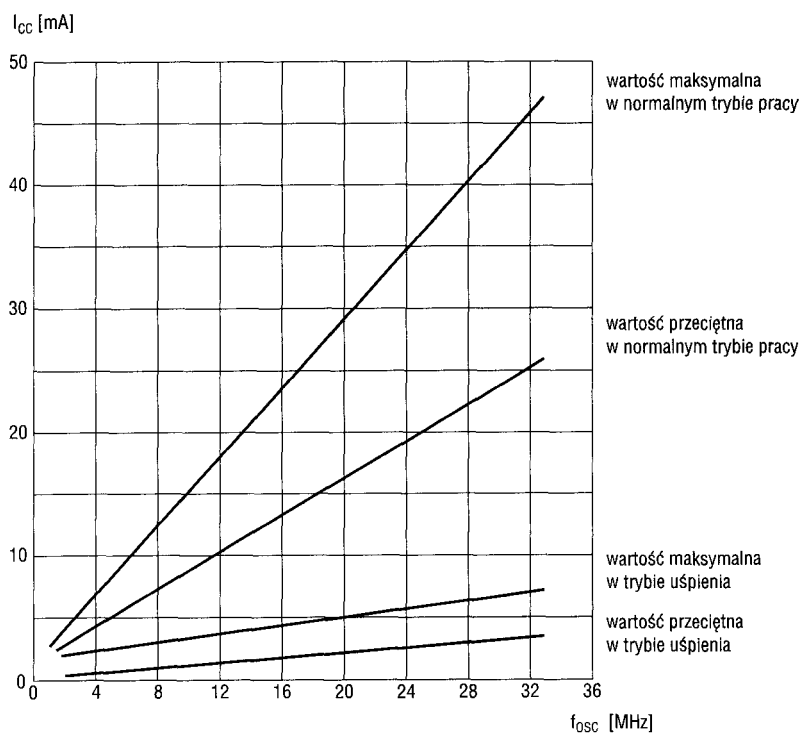
Mikrokontrolery rodziny '51 wykonane w technologii CMOS cechuje prawie proporcjonalna zależność prądu zasilania od napięcia zasilającego (rys. 5.2) i częstotliwości taktowania (rys. 5.3). Obie te zależności można, przynajmniej w pewnym zakresie, wykorzystać do zmniejszenia poboru mocy.

Tolerancja napięcia zasilającego mikrokontrolerów rodziny '51 wykonanych w technologii CMOS wynosi co najmniej $\pm 10\%$, a dla znacznej części mikrokontrolerów tolerancja ta wynosi $\pm 20\%$ lub jest jeszcze szersza (np. 2,7...6,0 V). Podane przez producenta tolerancje napięcia zasilającego stanowią przy tym nie tyle ograniczenie zakresu poprawnej pracy mikrokontrolera, ile raczej zakresu, w którym producent gwarantuje utrzymanie podanych przez niego danych katalogowych, a przede wszystkim parametrów czasowych, które wyraźnie pogarszają się ze spadkiem napięcia zasilania. Oznacza to, że mikrokontroler o 10-procentowej tolerancji napięcia zasilającego powinien bez większych problemów pracować przy napięciu zasilania 4,0 V (co stanowi 20-procentową odchyłkę od wartości 5,0 V), należy się natomiast wtedy liczyć np. z możliwością spadku maksymalnej częstotliwości pracy mikrokontrolera. Jest to istotne, ponieważ obniżenie napięcia zasilania z 5,0 V do 4,0 V pozwala w niektórych przypadkach zredukować prąd zasilania nawet o jedną trzecią. Wykorzystywanie układów do pracy w warunkach wykraczających poza nominalny zakres pracy podany przez producenta z pewnością nie należy jednak do dobrej praktyki inżynierskiej i jeśli już tego rodzaju rozwiązania brane są pod uwagę, to należy je stosować w ostateczności i tylko w takich urządzeniach, których ewentualne błędne działanie nie będzie przyczyną powstania szkód, zagrożeń itp. Dość często okazuje się zresztą, że zamiast stosować dany mikrokontroler na granicy zakresu jego poprawnej pracy, można wybrać inny mikrokontroler o zbliżonych możliwościach funkcjonalnych, a szerszym zakresie pracy, czy nawet ten

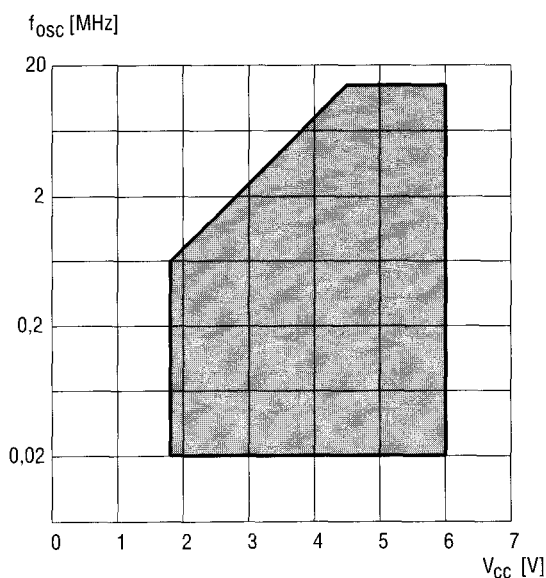


Rys. 5.2. Zmiany prądu zasilania w funkcji napięcia zasilającego na przykładzie mikrokontrolera CL410 pracującego z różnymi częstotliwościami taktowania: a), b) podczas normalnej pracy; c) w trybie uśpienia; d) w trybie zamrożenia

sam typ mikrokontrolera, ale wykonany przez innego producenta (np. gwarantującego pracę układu w szerszym zakresie napięć zasilających). Dobrym przykładem mogą tu być mikrokontrolery firmy OKI, których standardowe wykonania przeznaczone są do pracy przy napięciu zasilania 2,5...6,0 V. Zawsze jednak należy pamiętać o tym, że zmniejszenie napięcia zasilającego może (niekiedy istotnie) ograniczać maksymalną szybkość pracy mikrokontrolera (rys. 5.4), tzn. producent układu może np. w danych katalogowych określić, że dany mikrokontroler jest przeznaczony do pracy z częstotliwościami zegarowymi do 16 MHz i napięciem zasilania w zakresie 2,5...5,5 V, ale przy napięciu zasilania poniżej 4,5 V maksymalna częstotliwość zegarowa, przy której gwarantowana jest poprawna praca układu wynosi już tylko 12 MHz.



Rys. 5.3. Zmiana prądu zasilania w funkcji częstotliwości taktowania na przykładzie mikrokontrolera C51



Rys. 5.4. Obszar poprawnej pracy i zależność maksymalnej częstotliwości pracy mikrokontrolera od napięcia zasilania na przykładzie mikrokontrolera CL410

Tam, gdzie nie jest potrzebna maksymalna szybkość pracy mikrokontrolera, redukcję poboru mocy można uzyskać zmniejszając częstotliwość pracy. Ograniczeniem jest wówczas minimalna częstotliwość pracy mikrokontrolera, która najczęściej przyjmuje wartości z zakresu 0,5...1,2 MHz, ale niekiedy może wynosić nawet kilka MHz. Obecnie coraz częściej są oferowane mikrokontrolery o konstrukcji przystosowanej do pracy statycznej (np. mikrokontrolery firmy Atmel lub nowsze mikrokontrolery firmy Philips). Mikrokontrolery te mogą pracować z zegarem o dowolnie małej częstotliwości, tyle że poniżej pewnej wartości (np. 30 kHz) wymagają zastosowania zewnętrznego generatora taktującego. Warto w tym miejscu zauważyć, że np. jednoczesne obniżenie częstotliwości taktowania do wartości 1,2 MHz i napięcia zasilania do 3,0 V, pozwala na uzyskanie prądu zasilania około 0,5 mA w aktywnym trybie pracy mikrokontrolera (rys. 5.2b). Przykład ten wyraźnie wskazuje jak duże możliwości redukcji poboru mocy leżą w rozwiązaniach innych niż stosowanie trybów pracy ze zmniejszonym poborem mocy. Nic też nie stoi na przeszkodzie, by wszystkie te możliwości wykorzystywać równocześnie.

Tab. 5.2. Parametry pracy wybranych mikrokontrolerów rodziny '51. Wartość prądu zasilania mikrokontrolera w aktywnych trybach pracy określono dla $f_{OSC} = 12 \text{ MHz}$

Typ	Napięcie zasilania [V]		Maksymalny prąd zasilania [mA]			Przerwania umożliwiające wyjście ze stanu zamrożenia
	zakres V_{CC}	wartość min. V_{CC} w stanie zamrożenia	w zwykłym trybie pracy	w stanie uśpienia	w stanie zamrożenia	
C51	2,7...6,0*	2,0	20	5	0,075	INT0, INT1**
LV51	2,5...6,0	2,0	20	5	0,100	
S51	4,0...5,5	2,0	24	4	0,075	INT0, INT1
S52	4,0...5,5	2,0	25	4	0,075	INT0, INT1
S53	4,0...6,0	2,0	25	8	0,100	INT0, INT1
LS53	2,7...6,0	2,0	25	8	0,100	INT0, INT1
C52	4,5...5,5*	2,0	19	5	0,050	INT0, INT1**
LV52	2,5...6,0	2,0	25	7	0,100	
C54	4,5...5,5*	2,0	26	4	0,075	INT0, INT1
C55	4,5...5,5	2,0	26	7	0,100	INT0, INT1
C58	4,5...5,5*	2,0	26	8	0,100	INT0, INT1
C1051	3,0...6,0	2,0	15	5	0,100	
C1051U	3,0...6,0	2,0	15	5	0,100	
C2051	3,0...6,0	2,0	15	5	0,100	
C4051	3,0...6,0	2,0	15	5	0,100	
LS8252	2,7...6,0	2,0	25	7	0,100	INT0, INT1
S8252	4,0...6,0	2,0	25	7	0,100	INT0, INT1

* zależnie od producenta układu niektóre parametry, a zwłaszcza zakres napięć zasilania, mogą ulegać znacznym zmianom

** w najnowszych wersjach układów firmy Philips

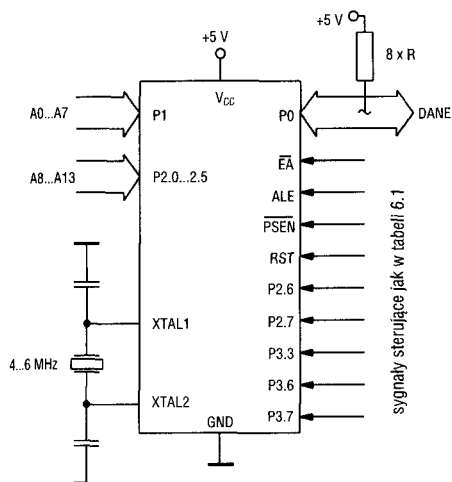
**Wewnętrzna pamięć
programu**



Każdy typ mikrokontrolera rodziny '51 może być wykonywany jako układ z wewnętrzną pamięcią programu, o zawartości określonej maską w trakcie produkcji. Opracowanie nowej maski jest jednak kosztowne, toteż stosowanie takich mikrokontrolerów staje się opłacalne dopiero w przypadku produkcji masowej liczonej co najmniej w tysiącach sztuk. Znacznie tańszym rozwiązaniem jest więc często wykorzystanie mikrokontrolera pracującego z zewnętrzną pamięcią programu. Rozwiązanie to ma jednak kilka istotnych wad. Po pierwsze porty P0 i P2 mikrokontrolera muszą wówczas pełnić funkcje alternatywne, co ogranicza istotnie liczbę wyprowadzeń, jakie mogą być wykorzystywane jako wejścia/wyjścia. Z drugiej strony stosowanie zewnętrznej pamięci programu uniemożliwia zabezpieczenie programu przed kopiowaniem, zwiększa rozmiar całego układu i moc pobieraną przez układ, pogarsza niezawodność układu itp. Wreszcie struktura wewnętrzna niektórych mikrokontrolerów rodziny '51 (np. C2051) z założenia uniemożliwia stosowanie zewnętrznej pamięci programu. Dość wygodną alternatywą pozostaje wówczas zastosowanie mikrokontrolera z wewnętrzną pamięcią programu typu EPROM. Z punktu widzenia działania mikrokontrolery te nie różnią się niczym od mikrokontrolerów z pamięcią programu, której zawartość ustalana jest w fazie produkcji. Jednocześnie pamięć EPROM mikrokontrolerów może być zaprogramowana dowolnym kodem. Wadą mikrokontrolerów z wewnętrzną pamięcią programu typu EPROM jest ich relatywnie wysoka cena – zwykle kilkakrotnie wyższa od układów przeznaczonych do pracy z pamięcią zewnętrzną. Ponadto nie wszystkie typy mikrokontrolerów rodziny '51 mają swoje EPROM-owe odmiany. Jeśli jednak dany typ mikrokontrolera jest produkowany w odmianie z wewnętrzną pamięcią EPROM, to z reguły jest on oferowany w dwóch rodzajach obudowy – z okienkiem lub bez. Mikrokontrolery w obudowie z okienkiem są droższe, ale obudowa ta umożliwia wymazywanie zapisanego programu (przez naświetlanie struktury układu promieniowaniem ultrafioletowym) i w konsekwencji testowanie kolejnych wersji programu za pomocą tego samego układu. Brak okienka uniemożliwia oczywiście wymazanie programu, w związku z czym mikrokontrolery z pamięcią EPROM w obudowie bezokienkowej umożliwiają tylko jednokrotne zaprogramowanie, stąd też często stosowana nazwa OTP (*One Time Programmable*). Zaletą mikrokontrolerów w wersji OTP jest ich cena (koszt obudowy ceramicznej z okienkiem jest wyraźnie wyższy od obudowy plastikowej), dlatego też są one przeważnie stosowane w małoseryjnej produkcji przetestowanych już układów. Odmianą pamięci typu EPROM jest też, coraz częściej stosowana w mikrokontrolerach jednoukładowych, pamięć FEEPROM (Flash). Zawartość takiej pamięci może być wymazywana elektrycznie, dzięki czemu nie musi ona mieć obudowy z okienkiem. Jednocześnie możliwość programowego zapisu i wymazywania zawartości takiej pamięci pozwala na stosunkowo prostą realizację systemów, w których pamięć programu może być programowana także w uruchamianym systemie.

6.1. Pamięć programu typu EPROM

Algorytm programowania wewnętrznej pamięci EPROM mikrokontrolera i wartość napięcia programującego V_{PP} mogą różnić się w zależności od producenta i typu układu. Jednakże większość produkowanych obecnie mikrokontrolerów rodziny '51 wykonywanych w technologii CMOS może być programowana za pomocą opisanego dalej algorytmu Quick lub jego szybszej odmiany – tzw. ulepszonego algorytmu Quick (*Improved Quick Pulse Programming Algorithm*). Układ, w jakim odbywa się



Rys. 6.1. Układ do programowania wewnętrznej pamięci programu typu EPROM mikrokontrolerów '51

programowanie mikrokontrolerów przedstawiono na rys. 6.1. Programowany mikrokontroler powinien być taktowany sygnałem o częstotliwości 4...6 MHz. Taktowanie jest konieczne, ponieważ podczas programowania mikrokontroler wykonuje wewnętrzne operacje przesyłania adresów i danych.

Pamięć EPROM jest programowana po jednym bajcie, przy czym adres programowanego bajtu pamięci jest określany stanem linii portów P1 i P2, a w przypadku pamięci o pojemności przekraczającej 16 KB także portu P3 (tab. 6.1). Wartość programowanego bajtu kodu jest podawana na port P0, a impulsy programujące na wyprowadzenie ALE mikrokontrolera. Zaprogramowanie bajtu następuje w wyniku wysłania odpowiedniej liczby impulsów programujących. Dla zwykłego algorytmu Quick liczba

Tab. 6.1. Sygnały sterujące podczas operacji programowania i weryfikacji pamięci EPROM mikrokontrolerów rodziny '51

Operacja	RST	PSEN	ALE	EA	P2.7	P2.6	P2.5...P2.0	P3.7	P3.6	P3.5	P3.4	P3.3	P0	P1
Programowanie bajtu kodu	1	0	ip	V _{pp}	1	0	A13...8	1	1	A15	A14	1	dane	A7...0
Weryfikacja bajtu kodu	1	0	1	1	0	0	A13...8	1	1	A15	A14	0	dane	A7...0
Programowanie tabeli szyfrującej	1	0	ip	V _{pp}	1	0	A13...8	1	0			1	dane	A7...0
Programowanie bitu zabezpieczającego nr 1	1	0	ip	V _{pp}	1	1	A13...8	1	1			1	dane	A7...0
Programowanie bitu zabezpieczającego nr 2	1	0	ip	V _{pp}	1	1	A13...8	0	0			1	dane	A7...0
Programowanie bitu zabezpieczającego nr 3	1	0	ip	V _{pp}	0	1	A13...8	0	1			1	dane	A7...0
Odczyt sygnatury	1	0	1	1	0	0	A13...8	0	0			0	dane	A7...0

Uwagi:

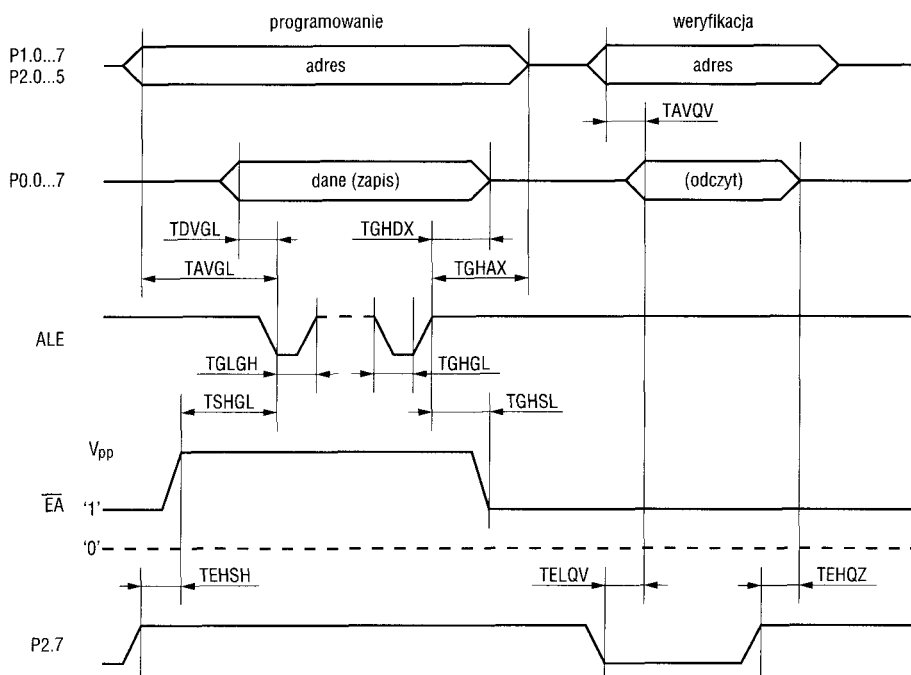
- 1) ip – impulsy programujące o zależnościach czasowych jak na rys. 6.2.
- 2) Liczba wyprowadzeń wykorzystywanych do adresowania programowanego bajtu zależy od pojemności pamięci EPROM.
- 3) Poziom logiczny sygnał jaki należy wymusić na wyprowadzeniu RST zależy oczywiście od aktywnego poziomu sygnału zerowania. W tabeli wpisano wartość „1” typową dla większości mikrokontrolerów rodziny '51 (o aktywnym wysokim poziomie sygnału RST).
- 4) W przypadku niektórych mikrokontrolerów (o aktywnym niskim poziomie sygnału RST – np. C575, C576) przejście do operacji programowania następuje w wyniku wymuszenia na wyprowadzeniu PSEN stanu wysokiego w momencie włączenia zasilania, a następnie zmiany stanu wyprowadzenia PSEN na niski przy aktywnym stanie wyprowadzenia RST.

UWAGA

Należy zwracać baczna uwagę, by napięcie programujące V_{pp} było pozbawione jakichkolwiek, chwilowych nawet przepięć, ponieważ nawet niewielkie, krótkotrwałe przekroczenie katalogowej wartości napięcia V_{pp} może spowodować trwałe uszkodzenie programowanego układu.

impulsów programujących wynosi 25, co przy minimalnym dozwolonym czasie trwania tych impulsów odpowiada programowaniu z szybkością około 0,33 KB/s. W przypadku ulepszonego algorytmu Quick bajty kodu można programować za pomocą zaledwie 5 impulsów programujących – 25 impulsów jest wymaganych natomiast przy programowaniu pozostałych danych (tabeli szyfrującej i bitów zabezpieczających). Takie zmniejszenie liczby impulsów programujących pozwala na zwiększenie szybkości programowania pamięci EPROM do około 1,5 KB/s. Przykładowe zależności czasowe sygnałów sterujących programowaniem i weryfikacją pamięci EPROM umieszczono na rys. 6.2 i w tab. 6.2.

Mikrokontrolery rodziny '51 z wewnętrzną pamięcią EPROM mają dwa rodzaje zabezpieczeń przed niepożądanym odczytem kodu – tabelę szyfrującą i bity zabezpieczające. Tabela szyfrująca składa się w zależności od mikrokontrolera z 16, 32 lub 64 bajtów. Przy weryfikacji zawartości pamięci EPROM mikrokontrolera, odczytywana wartość jest funkcją EX-OR-NOT rzeczywistego kodu umieszczonego w pamięci i odpowiedniego bajtu z tabeli szyfrującej. Numer użytego do tego celu bajtu tabeli szyfrującej jest określany jako wartość adresu odczytywanego bajtu kodu modulo długość



Rys. 6.2. Zależności czasowe sygnałów występujących podczas programowania i weryfikacji standardowej pamięci EPROM mikrokontrolerów rodziny '51

Tab. 6.2. Parametry sygnałów sterujących występujących podczas programowania i weryfikacji standardowej pamięci EPROM mikrokontrolerów rodziny '51 (dla algorytmu Quick)

Oznaczenie	Parametr	Min.	Maks.	Jednostki
V _{pp}	Napięcie programowania	12,5	13,0	V
I _{pp}	Pobór prądu podczas programowania		75	mA
1/t	Częstotliwość taktowania	4	6	MHz
TAVGL	Adres ważny do PROG(L)	48t		
TGHAX	PROG(H) do adres (3. stan)	48t		
TDVGL	Dane ważne do PROG(L)	48t		
TGHDX	PROG(H) do dane (3. stan)	48t		
TEHSH	Sygnały sterujące ważne do V _{pp}	48t		
TSHGL	V _{pp} do PROG(L)	10		μs
TGHSL	PROG(H) do końca V _{pp}	10		μs
TGLGH	Szerokość impulsów	90	110	μs
TAVQV	Adres ważny do dane ważne		48t	
TELQV	Sygnały sterujące ważne do dane ważne		48t	
TEHQZ	Zmiana sygnałów sterujących do dane (3. stan)	0	48t	
TGHGL	PROG(H) do PROG(L)	10		μs

tabeli szyfrującej. Oznacza to, że do szyfrowania jest wykorzystywany pierwszy bajt pamięci programu z pierwszym bajtem tabeli szyfrującej, drugi bajt pamięci, z drugim bajtem tabeli itd. Początkowo oraz po wymazaniu zawartości pamięci EPROM tabela szyfrująca zawiera wyłącznie bajty 0FFh, tak więc kod jest wówczas odczytywany w postaci niezaszyfrowanej ($n \oplus 0FFh = n$). Tabela szyfrująca jest programowana analogicznie jak pamięć programu, z tym że podczas programowania tabeli szyfrującej inny musi być stan wyprowadzenia P3.6 (patrz **tab. 6.1**). Zawartości tabeli szyfrującej nie można odczytać, a po jej zaprogramowaniu weryfikacja pamięci programu będzie dawała wyłącznie zaszyfrowaną informację.

Drugim rodzajem zabezpieczeń kodu programu w mikrokontrolerach rodziny '51 są tzw. bity zabezpieczające. Mikrokontroler może mieć do trzech takich bitów. Zaprogramowanie pierwszego z nich (patrz **tab. 6.1**) powoduje zablokowanie dalszego programowania pamięci programu i tabeli szyfrującej, w dalszym ciągu jest możliwe jednak programowanie pozostałych bitów zabezpieczających i weryfikacja pamięci programu. Zaprogramowanie drugiego bitu zabezpieczającego uniemożliwia odczytanie zawartości (weryfikację) pamięci programu. Po zaprogramowaniu trzeciego bitu zabezpieczającego program może być wykonywany wyłącznie z pamięci wewnętrznej.



W przypadku wykorzystywania tabeli szyfrującej należy jednak pamiętać o tym, że pozostawienie większego obszaru pamięci programu w stanie niezaprogramowanym spowoduje, że podczas weryfikacji w miejscach tych będzie odczytywana zawartość tabeli szyfrującej (bo $n \oplus 0FFh = n$). Z tego względu, dążąc do maksymalnego zabezpieczenia kodu przed niepożądanym odczytem, wszystkie niewykorzystywane obszary pamięci EPROM należy zaprogramować bajtami o różnej (niejednakowej) wartości.

Mikrokontrolery rodziny '51 mają zwykle dwa lub trzy tzw. bajty sygnatur. Bajty te umieszczone są pod adresami 30h, 31h i 60h i mogą być odczytane na podobnej zasadzie jak pamięć programu (**tab. 6.1**). Wartości bajtów sygnatur określają producenta i typ mikrokontrolera (np. wartości 15h i 90h identyfikują mikrokontroler 87C451 wyprodukowany przez firmę Philips). Bajty sygnatur mogą być wykorzystane przez układ programatora do automatycznego rozpoznawania programowanego układu.

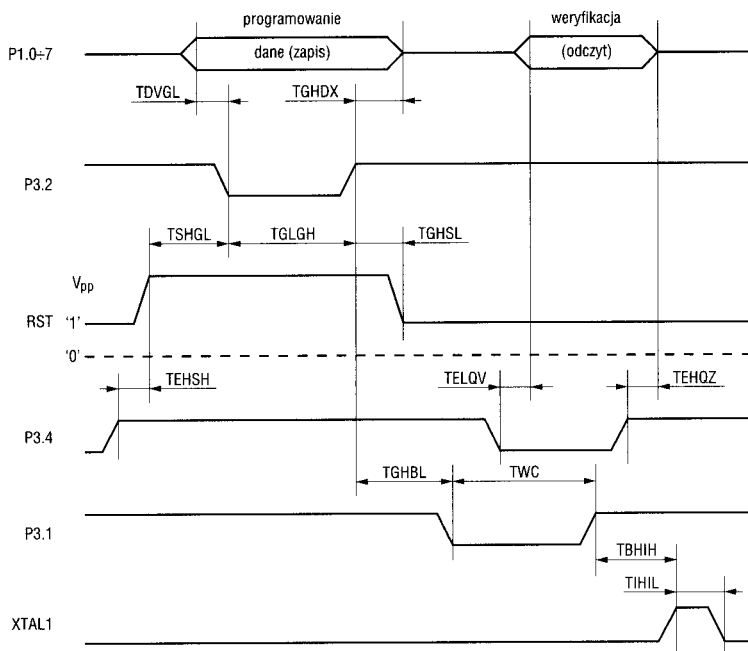
6.2. Pamięć programu typu Flash

Niektórzy producenci mikrokontrolerów jednoukładowych jako wewnętrzną pamięć programu stosują pamięć EEPROM typu Flash (w skrócie FEEPROM lub po prostu Flash). Główną zaletą wykorzystywania takiej pamięci jest możliwość modyfikowania jej zawartości bez konieczności wymontowywania mikrokontrolera z uruchamianego systemu. Jest to możliwe, ponieważ zawartość pamięci typu Flash (całej pamięci lub mniejszych jej bloków) jest wymazywana za pomocą impulsów elektrycznych. Niektóre mikrokontrolery wyposażone w pamięć Flash umożliwiają wymazywanie zawartości tej pamięci za pomocą pojedynczej operacji wyzwalanej programowo. Ponadto część pamięci Flash nie wymaga stosowania zewnętrznego napięcia programującego – np. 12,75 V, czy 21 V jak w przypadku zwykłych pamięci EPROM. Niektóre mikrokontrolery wyposażone w pamięć programu typu Flash mają też wbudowany program ładujący, który umożliwia programowanie i wymazywanie zawartości pamięci programu poprzez interfejs szeregowy.

Mikrokontrolery 89C51, 89LV51, 89C52, 89LV52, 89C55, 89S51, 89S52, 89S53, 89(L)S8252, 89C1051(U), 89C2051 i 89C4051 firmy Atmel wyposażono w wewnętrzną pamięć programu typu Flash. Pierwszych dziewięć z wymienionych mikrokontrolerów może być programowanych z wykorzystaniem algorytmu podobnego do tego, jaki stosowany jest w mikrokontrolerach z pamięcią EPROM. Trzy ostatnie mikrokontrolery mają zaledwie 20 wyprowadzeń i już choćby z tego powodu muszą one wykorzystywać inny algorytm programowania.

6.2.1. Programowanie pamięci Flash mikrokontrolerów o zmniejszonej liczbie wyprowadzeń

Ze względu na bardzo małą liczbę wyprowadzeń, sposób programowania pamięci Flash w mikrokontrolerach 89C1051(U), 89C2051 i 89C4051 musi istotnie różnić się od sposobu programowania pamięci EPROM np. w układach 87C51. Najważniejszą cechą omawianych mikrokontrolerów o zmniejszonej liczbie wyprowadzeń jest brak zewnętrznych linii, służących do adresowania pamięci Flash podczas jej programowania. Mikrokontrolery te wyposażone zostały w wewnętrzny licznik adresowy. Licznik ten jest zerowany narastającym zboczem sygnału na wyprowadzeniu RST oraz inkrementowany dodatnimi impulsami podawanymi na wyprowadzenie XTAL1 mikrokontrolera. Wykorzystanie wejścia XTAL1 do zwiększania zawartości wewnętrznego licznika adresów programowanej pamięci uniemożliwia, rzecz jasna, wykorzystanie wyprowadzeń XTAL1, XTAL2 do standardowego taktowania mikrokontrolera. Mikrokontroler musi zatem mieć wewnętrzny układ generujący znaczną część uzależnień czasowych wykorzystywanych w procesie programowania pamięci Flash. Z tego



Rys. 6.3. Zależności czasowe występujące podczas programowania i weryfikacji pamięci Flash w mikrokontrolerach 20-końcówkowych

samemu powodowi sposób sterowania mikrokontrolera podczas programowania pamięci jest dość precyzyjnie określony – poniżej podano algorytm programowania zalecany przez firmę Atmel. Przebiegi czasowe występujące podczas operacji programowania i weryfikacji zawartości pamięci Flash w mikrokontrolerach 89C1051(U), 89C2051 i 89C4051 pokazano na rys. 6.3, a wartości poszczególnych parametrów czasowych umieszczono w tab. 6.3.

Wprowadzenie mikrokontrolera w tryb programowania pamięci Flash następuje w wyniku załączenia napięcia zasilania przy niskim stanie wyprowadzeń RST i XTAL1 i pozostałych wyprowadzeniach niepodłączonych (pozostawionych na potencjale pływającym). Opisany stan wyprowadzeń musi być utrzymany przez co najmniej 10 ms od momentu załączenia napięcia zasilania. Po upływie 10 ms na wyprowadzenia RST i P3.2 należy podać stan wysoki (zmiana stanu wyprowadzenia RST spowoduje wyzerowanie licznika adresów), a następnie, w zależności od wykonywanej operacji, wymusić odpowiedni stan na wyprowadzeniach P3.3, P3.4, P3.5 i P3.7 (tab. 6.4).

Jeśli wykonywaną operacją ma być programowanie pamięci Flash, to na wyprowadzenia portu P1 należy podać bajt, jaki ma zostać zapisany w pamięci, a następnie podnieść napięcie na wejściu RST do wartości $12,0 \pm 0,5$ V (napięcie programujące) i wysłać ujemny impuls programujący na wyprowadzenie P3.2 mikrokontrolera. Czas trwania impulsu programującego musi mieścić się w zakresie 1...110 μ s (impuls musi mieć długość 10 ms jeśli wykonywana jest operacja wymazywania zawartości pamięci). Dla programowania pamięci Flash występującej w mikrokontrolerach firmy Atmel bardzo istotne jest jednak to, że wspomniany krótki impuls programujący służy w rzeczywistości jedynie wyzwoleniu operacji programowania kolejnego bajtu pamięci

Tab. 6.3. Parametry sygnałów sterujących programowaniem i weryfikacją pamięci Flash mikrokontrolerów o zmniejszonej liczbie wyprowadzeń

Oznaczenie	Parametr	Min.	Maks.	Jednostki
V _{pp}	Napięcie programowania	11,5	12,5	V
I _{pp}	Pobór prądu podczas programowania		0,25	mA
TDVGL	Dane ważne do P3.2 (L)	1,0		μs
TGHDX	P3.2 (H) do dane (3. stan)	1,0		μs
TEHSH	P3.4 (H) do V _{pp}	1,0		μs
TSHGL	V _{pp} do P3.2 (L)	10		μs
TGHSL	P3.2 (H) do końca V _{pp}	10		μs
TGLGH	Szerokość impulsów programujących na P3.2	1	110	μs
TELQV	P3.4 (L) do dane ważne		1,0	μs
TEHQZ	P3.4 (H) do dane (3. stan)	0	1,0	μs
TGHBL	P3.2 (H) do P3.1(L)		50	ns
TWC	Czas trwania cyklu zapisu do pamięci		2,0	ms
TBHIH	P3.1 (H) do XTAL1 (H)	1,0		μs
TIHIL	XTAL1(H) DO XTAL1 (L)	200		ns

(sterowanej w pełni przez specjalny wewnętrzny układ programujący). Rzeczywisty czas programowania pojedynczego bajtu pamięci jest znacznie dłuższy (nie przekracza jednak nigdy 2 ms; wartość typowa – około 1,2 ms). Różnica między czasem trwania zewnętrznego impulsu programującego i rzeczywistym czasem programowania wymusza stosowanie dodatkowych środków gwarantujących, że próba programowania następnego bajtu pamięci nie zostanie rozpoczęta przed zakończeniem wewnętrznej operacji programowania poprzedniego bajtu. Najprostszą możliwością jest oczywiście programowanie na najgorszy przypadek, czyli wysyłanie impulsów programujących nie częściej niż co 2 ms (stanowiące graniczną wartość rzeczywistego czasu programowania bajtu pamięci Flash). Rozwiązanie to może jednak znacznie wydłużyć czas programowania całej pamięci, ponieważ w wielu przypadkach operacja programowania bajtu zostanie zakończona znacznie wcześniej. Z tego względu konstruktorzy firmy Atmel przewidzieli dwa mechanizmy umożliwiające testowanie stanu wewnętrznego

Tab. 6.4. Sygnały sterujące podczas operacji programowania i weryfikacji pamięci Flash mikrokontrolerów o zmniejszonej liczbie wyprowadzeń

Operacja	RST	P3.2	P3.3	P3.4	P3.5	P3.7	P1
Programowanie bajtu kodu	V _{pp}	ip	0	1	1	1	dane
Weryfikacja bajtu kodu	1	1	0	0	1	1	dane
Wymazywanie pamięci	V _{pp}	ip	1	0	0	0	
Programowanie bitu zabezpieczającego nr 1	V _{pp}	ip	1	1	1	1	
Programowanie bitu zabezpieczającego nr 2	V _{pp}	ip	1	1	0	0	
Odczyt sygnatury	1	1	0	0	0	0	dane

ip – impulsy programujące (przy wymazywaniu pamięci impuls 10 ms)

układu programowania pamięci Flash. Pierwszy z nich polega na powtarzaniu odczytu (weryfikacji) ostatnio programowanego bajtu pamięci. Operację programowania danego bajtu można uznać za zakończoną dopiero w momencie, gdy próba odczytu bajtu pamięci programu spod ostatnio programowanego adresu da wynik identyczny z wartością wpisaną za pomocą ostatniego impulsu programującego. W celu przeprowadzenia operacji weryfikacji bezpośrednio po próbie zaprogramowania danego bajtu pamięci Flash wystarczy obniżyć napięcie na wejściu RST z 12 V do poziomu stanu wysokiego oraz podać niski stan na wyprowadzenie P3.4 i wysoki na P4.7. Odczytywany bajt pamięci programu pojawi się wówczas na wyprowadzeniach portu P1. Drugą możliwością określenia końca operacji programowania bajtu pamięci Flash jest sprawdzanie stanu wyprowadzenia P3.1 mikrokontrolera. Podczas cyklu zapisu bajtu pamięci programu wyprowadzenie to jest ustawiane w stan niski tuż po narastającym zboczu sygnału P3.2 (maksymalnie w ciągu 50 ns). Koniec wewnętrznej operacji programowania jest sygnalizowany powrotem wyprowadzenia P3.1 do stanu wysokiego. Po stwierdzeniu, że operacja programowania danego bajtu pamięci Flash została zakończona, zawartość wewnętrznej licznika adresów może być zwiększona przez wysłanie impulsu na wyprowadzenie XTAL1, po czym operacja programowania (i weryfikacji) może być powtórzona dla kolejnego bajtu pamięci Flash.

Programowanie bajtu pamięci Flash występującej w mikrokontrolerach firmy Atmel możliwe jest tylko wtedy, gdy przed rozpoczęciem operacji programowania bajt ten ma wartość 0FFh. W przeciwnym razie przed przystąpieniem do operacji programowania zawartość pamięci musi zostać wymazana (skasowana). Należy przy tym pamiętać, że podobnie jak w przypadku standardowej pamięci EPROM, zawartość pamięci Flash w mikrokontrolerach firmy Atmel może być wymazana wyłącznie w całości (wyjątkiem jest pamięć mikrokontrolerów wyposażonych w interfejs SPI i programowana w trybie szeregowym).



Odłączenie napięcia zasilającego po zakończeniu wybranych operacji na pamięci Flash powinno być poprzedzone wymuszeniem niskiego stanu na wyprowadzeniach XTAL1 i RST i pozostawieniem reszty wyprowadzeń mikrokontrolera niepodłączonych (na potencjale pływającym).

6.2.2. Programowanie pamięci Flash pozostałych mikrokontrolerów firmy Atmel w odmianach C i LV

Mikrokontrolery 89C51, 89LV51, 89C52, 89LV52 i 89C55 firmy Atmel oferowane są z dwiema wersjami pamięci Flash, różniącymi się napięciem programowania (12 V lub 5 V). Zaletą pierwszej z tych wersji (12 V) jest możliwość programowania przy użyciu standardowych programatorów mikrokontrolerów z pamięcią EPROM, zaletą drugiej (5 V) – łatwa realizacja programowania wewnątrzukładowego. Sposób programowania pamięci Flash omawianych mikrokontrolerów jest bardzo zbliżony do programowania standardowej wewnętrznej pamięci programu typu EPROM (rozdz. 6.1), dlatego też dalej omówiono jedynie różnice dotyczące sposobu programowania obu tych typów pamięci.

Pierwsza z różnic polega na tym, że do zaprogramowania bajtu kodu lub wymazania pamięci Flash wystarcza pojedynczy impuls podany na wyprowadzenie ALE (PROG). Czas trwania tego impulsu powinien wynosić 1...110 μ s przy programowaniu i 10 ms w przypadku operacji wymazywania zawartości pamięci. Pozostałe parametry czasowe są analogiczne jak przy programowaniu mikrokontrolerów ze standardową pamięcią EPROM. Podobnie jak w przypadku mikrokontrolerów o zmniejszonej liczbie wyprowadzeń, krótki impuls programujący służy jedynie wyzwoleniu operacji programowania kolejnego bajtu pamięci (czas programowania pojedynczego bajtu pamięci wynosi zwykle około 1,5 ms – nie przekracza nigdy 2 ms). Mechanizmy umożliwiające testowanie stanu wewnętrznego układu programowania pamięci Flash występujące w omawianych mikrokontrolerach są niemal identyczne, jak w opisanych wcześniej mikrokontrolerach o zmniejszonej liczbie wyprowadzeń. Pierwszy z nich polega na powtarzaniu weryfikacji ostatnio programowanego bajtu pamięci – operacja programowania jest zakończona, jeśli weryfikacja daje wynik pozytywny. Drugą możliwością jest sprawdzanie stanu wyprowadzenia P3.4 mikrokontrolera. Podczas cyklu zapisu bajtu pamięci programu wyprowadzenie to jest ustawiane w stan niski tuż po narastającym zboczu sygnału ALE (maksymalnie w ciągu 1 μ s). Koniec wewnętrznej operacji programowania sygnalizowany jest powrotem wyprowadzenia P3.4 do stanu wysokiego.

Druga z różnic w programowaniu pamięci Flash występującej w mikrokontrolerach firmy Atmel i standardowej pamięci EPROM dotyczy napięcia programującego. Jeśli programowany jest mikrokontroler w wersji z napięciem programującym 12 V, to rzeczywiste napięcie programujące musi mieścić się w zakresie 11,5...12,5 V. W przypadku programowania pamięci Flash w wersji 5-woltowej na wyprowadzenie \overline{EA} nie jest przykładane żadne napięcie programujące – wyprowadzenie to jest jedynie utrzymywane w stanie wysokim.

Kolejna z omawianych różnic dotyczy maksymalnej częstotliwości taktowania mikrokontrolerów podczas programowania pamięci Flash. W mikrokontrolerach firmy Atmel częstotliwość ta ograniczona jest do maksymalnej wartości częstotliwości zegarowej danego mikrokontrolera, podczas gdy w mikrokontrolerach ze standardową pamięcią EPROM zakres częstotliwości taktowania podczas operacji programowania wynosi 4...6 MHz.

Ostatnia z różnic wiąże się ze sposobem kasowania pamięci Flash. W celu przeprowadzenia operacji wymazywania zawartości tej pamięci (nie występującej w mikrokontrolerach ze standardową pamięcią EPROM i z tego powodu nie ujętej w **tab. 6.1**) wyprowadzenie P2.6 mikrokontrolera należy ustawić w stan wysoki, a na wyprowadzeniach P2.7, P3.6 i P3.7 wymusić stan niski. Podobnie jak w przypadku standardowej pamięci EPROM, zawartość pamięci Flash w mikrokontrolerach firmy Atmel może być wymazana wyłącznie w całości (wyjątkiem jest pamięć mikrokontrolerów wyposażonych w interfejs SPI i programowana w trybie szeregowym).

6.2.3. Programowanie równoległe pamięci Flash mikrokontrolerów 89S51, S52, S53, S8252

Mikrokontrolery 89S51, 89S52, 89S53, 89S8252 są pod wieloma względami układami bardzo zbliżonymi do mikrokontrolerów 89C52 i 89LV52, jednakże istotnymi elementami układów 89S51, 89S52, 89S53, 89S8252 nie występującymi we wspomnianych wcześniej mikrokontrolerach jest interfejs SPI, zaś w przypadku układu 89S8252

także pamięć danych typu EEPROM. Występowanie tych elementów zmienia nieco możliwości programowania tych mikrokontrolerów.

O ile mikrokontrolery 89C51, 89LV51 itp. oferowane są w dwóch wersjach (przystosowanych do napięcia programującego o wartości 12 V lub 5 V), to układy 89S51, 89S52, 89S53 i 89S8252 tylko w jednej. Jednakże pamięć Flash mikrokontrolerów 89S51, 89S52, 89S53 i 89S8252 może być programowana zarówno z wykorzystaniem programatora (stosując napięcie programowania 12 V), jak i bezpośrednio w uruchamianym układzie (wykorzystując napięcie programowania 5 V). W tym drugim przypadku programowanie odbywa się za pośrednictwem interfejsu SPI. Inną istotną cechą pamięci Flash mikrokontrolerów 89S8252 jest to, że nie jest to wyłącznie pamięć programu – część jej struktury pełni funkcję pamięci danych typu EEPROM. Biorąc pod uwagę, że pamięć programu (8 KB) i pamięć danych typu EEPROM (2 KB) tworzą fizycznie jeden element wewnętrznej struktury mikrokontrolera (pamięć Flash o pojemności 10 KB), konstruktorzy firmy Atmel wyposażyli układ 89S8252 w pewne mechanizmy, które umożliwiają programowanie pamięci danych typu EEPROM w sposób analogiczny do programowania pamięci programu.

W przypadku korzystania z programatora, sposób programowania pamięci Flash układów 89S51, 89S52, 89S53 i 89S8252 jest podobny, jak dla 12-woltowych odmian mikrokontrolerów 89C52, 89LV52 itp., jednakże sposób sterowania poszczególnych wprowadzeń mikrokontrolera 89S8252 podczas programowania równoległego jest nieco

Tab. 6.5. Sygnały sterujące podczas operacji programowania i weryfikacji pamięci Flash mikrokontrolerów 89S51, 89S52, 89S53 i 89S8252 (programowanie równoległe)

Operacja	RST	PSEN	ALE	EA	P2.7	P2.6	P2.5...P2.0	P3.7	P3.6	P0	P1
Programowanie bajtu pamięci	1	0	ip	V_{pp}	1	0	A13...8	1	1	dane do zapisu	A7...0
Weryfikacja bajtu pamięci	1	0	1	V_{pp}	0	0	A13...8	1	1	dane odczytywane	A7...0
Wymazywanie pamięci EEPROM programu (i danych)	1	0	ip	V_{pp}	0	1		0	0		
Programowanie bitów zabezpieczających	1	0	ip	V_{pp}	0	1		0	1	bit 1: P0.7 = 0 bit 2: P0.6 = 0 bit 3: P0.5 = 0	
Odczyt stanu bitów zabezpieczających	1	0	1	V_{pp}	1	1		0	0	bit 1: P0.2 bit 2: P0.1 bit 3: P0.0	
Odczyt bitu blokującego programowanie szeregowo	1	0	1	V_{pp}	1	1		1	0	bit: P0.0	
Odblokowanie programowania szeregowo	1	0	ip	V_{pp}	1	0		1	0	P0.0 = 0	
Zablokowanie programowania szeregowo	1	0	ip	V_{pp}	1	0		1	0	P0.0 = 1	
Odczyt sygnatur	1	0	1	V_{pp}	0	0	A13...8	0	0	sygnatura	A7...0

1) ip – impulsy programujące (w przypadku wymazywania pamięci oraz odblokowywania lub blokowania programowania szeregowo impuls 10 ms, w pozostałych przypadkach 1...110 μ s).

2) Podczas programowania linia P3.4 funkcjonuje jako RDY/BSY.

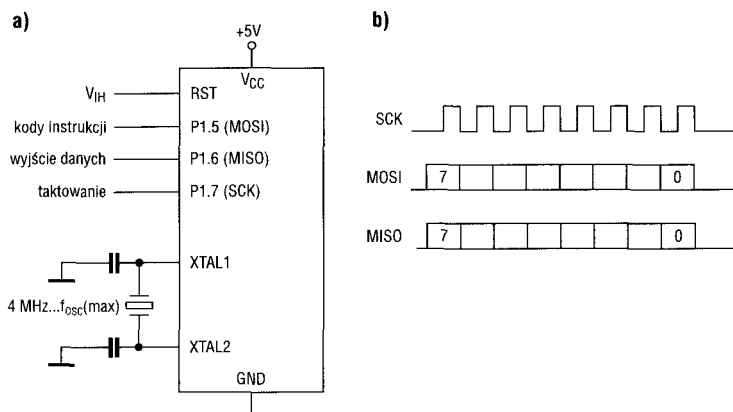
inny, niż w przypadku pozostałych mikrokontrolerów (tab. 6.5). Pamięć danych typu EEPROM programowana jest identycznie jak pamięć programu, a rozróżnienie tych pamięci jest dokonywane przez umieszczenie pamięci danych typu EEPROM w obszarze adresowym 2000h...27FFh, podczas gdy pamięć programu zajmuje obszar 0000h...1FFFh.

6.2.4. Programowanie szeregowe pamięci Flash mikrokontrolerów 89S51, S52, S53, S8252

Pamięć Flash mikrokontrolerów 89S51, 89S52, 89S53 i 89S8252 może być również programowana za pomocą interfejsu SPI. Podczas programowania szeregowego wyprowadzenie RST musi znajdować się w stanie wysokim, a częstotliwość zegarowa mikrokontrolera (ustalana za pomocą rezonatora kwarcowego dołączonego do wyprowadzeń XTAL lub uzyskiwana z zewnętrznego generatora sygnału taktującego) musi mieścić się w zakresie od 4 MHz do maksymalnej częstotliwości pracy danego mikrokontrolera (rys. 6.4). Podczas programowania szeregowego interfejs SPI programowanego mikrokontrolera pracuje w trybie urządzenia podrzędnego (patrz rozdz. 3.3.5), a częstotliwość sygnału taktującego przesyłanego linią SCK nie może przekraczać $1/40$ częstotliwości zegarowej mikrokontrolera (np. dla $f_{OSC} = 12$ MHz musi zachodzić $f_{SCK} \leq 300$ kHz).

Programowanie szeregowe mikrokontrolera 89S8252 (podobnie jak i równoległe) umożliwia ustalenie nowej zawartości zarówno pamięci programu, jak i pamięci danych typu EEPROM. Podczas programowania szeregowego zasoby te podzielone są na rozłączne przestrzenie adresowe (przestrzeń adresową pamięci programu i przestrzeń pamięci danych) i są programowane za pomocą różnych komend (tab. 6.6). Pamięć programu zajmuje obszar adresowy 0000h...1FFFh, zaś pamięć danych 000h...7FFh.

Wprowadzenie mikrokontrolerów 89S51, 89S52, 89S53 i 89S8252 w tryb szeregowego programowania pamięci Flash następuje w wyniku załączenia napięcia zasilania, ustawienia wyprowadzenia RST w stan wysoki i podłączenia zewnętrznego źródła sygnału zegarowego (tylko wtedy, gdy nie jest używany generator wewnętrzny, wykorzystujący rezonator podłączony do wyprowadzeń XTAL). Następnie, po przerwie co najmniej 10 ms, należy wysłać interfejsem SPI komendę rozpoczęcia programowania



Rys. 6.4. Uproszczony schemat ilustrujący sposób programowania pamięci programu typu Flash w mikrokontrolerze 89S8252: a) funkcje linii interfejsu SPI; b) sposób przesyłania danych

szeregowego. Komenda ta nie odniesie skutku, jeśli ustawiony będzie specjalny bit zabezpieczający przed programowaniem szeregowym. Bit ten jest zerowany w procesie produkcji – później może być on zerowany i ustawiany w wyniku wykonywania operacji zezwolenia i blokowania programowania szeregowego, dostępnych jedynie w trybie programowania równoległego (tab. 6.5). Po pomyślnym wykonaniu komendy rozpoczęcia programowania szeregowego można przejść do właściwej procedury programowania. Po zakończeniu programowania stan wyprowadzenia RST może być zmieniony na niski – mikrokontroler rozpocznie wówczas wykonywanie programu jak po zwykłej operacji zerowania.

Wszystkie operacje związane z programowaniem pamięci Flash mikrokontrolerów 89S53 i 89S8252 są wykonywane w odpowiedzi na, przesyłane za pomocą interfejsu SPI, 3-bajtowe komendy (tab. 6.6). W przypadku nowszych 89S51 i 89S52 komendy są 4-bajtowe (tab. 6.7). Kody komend oraz wartości adresu itp. są przesyłane linią MOSI – wyjątkiem są komendy odczytu bajtu pamięci, podczas których wartość odczytywanego bajtu jest zwracana za pośrednictwem linii MISO.

Charakterystyczną cechą szeregowego programowania mikrokontrolerów 89S51, 89S52, 89S53 i 89S8252 jest to, że każdy programowany bajt pamięci jest automatycznie wymazywany przed zapisem. Dzięki temu nie zachodzi konieczność wymazywania zawartości całej pamięci programu i danych w razie potrzeby zmiany zawartości tylko niewielkiego jej fragmentu – należy jednak pamiętać, że jest to zasada obowiązująca wyłącznie w trybie programowania szeregowego. Podczas programowania szeregowego zawartość dowolnego bajtu pamięci Flash może być zmodyfikowana (przeprogramowa-

Tab. 6.6. Wykaz komend szeregowego programowania pamięci Flash mikrokontrolerów 89(L)S53 i 89(L)S8252

Operacja	Format komendy (3 bajty)	Opis
Komenda rozpoczęcia programowania szeregowego	10101100 01010011 XXXXXXXX	Wprowadza mikrokontroler w tryb programowania szeregowego, jeśli wyprowadzenie RST jest w stanie wysokim
Wymazywanie pamięci	10101100 XXXXX100 XXXXXXXX	Kasuje zawartość pamięci programu i pamięci danych typu EEPROM. Czas trwania operacji ok. 16 ms.
Odczyt pamięci programu	AAAAAY01 młodszy bajt adresu XXXXXXXX	Odczytuje zawartość wybranego bajtu pamięci programu. Wartość odczytywanego bajtu jest przesyłana linią MISO podczas transmisji trzeciego bajtu komendy. AAAAA – 5 starszych bitów adresu (A12...8) Y = 0 dla (L)S8252, Y=A13 dla (L)S53
Programowanie pamięci programu	AAAAAY10 młodszy bajt adresu bajt do zapisu	Programuje wybrany bajt pamięci programu. AAAAA – 5 starszych bitów adresu (A12...8) Y = 0 dla (L)S8252, Y=A13 dla (L)S53
Odczyt pamięci danych tylko (L)S8252	00AAA101 młodszy bajt adresu XXXXXXXX	Odczytuje zawartość wybranego bajtu pamięci danych. Wartość odczytywanego bajtu jest przesyłana linią MISO podczas transmisji trzeciego bajtu komendy. AAA – 3 starsze bity adresu (A10...8)
Programowanie pamięci danych tylko (L)S8252	00AAA110 młodszy bajt adresu bajt do zapisu	Programuje wybrany bajt pamięci danych. AAA – 3 starsze bity adresu (A10...8)
Programowanie bitów zabezpieczających	10101100 BBBXX111 XXXXXXXX	Programuje bity zabezpieczające. BBB – wartość bitów zabezpieczających, począwszy od bitu nr 1 (B1...3).

na) i odczytana (np. w celu weryfikacji). Możliwe jest również programowanie bitów zabezpieczających, nie da się natomiast odczytać stanu tych bitów – może on być określony jedynie pośrednio – przez obserwację, jakie funkcje zabezpieczające zostały uaktywnione. Jak wspomniano, przy programowaniu szeregowym nie zachodzi konieczność wymazywania zawartości całej pamięci Flash, jednakże wykonanie takiej operacji jest możliwe – powoduje ono wymazanie całej pamięci Flash, tj. zarówno pamięci programu, jak i pamięci danych typu EEPROM. Wymazywanie trwa około 16 ms i powoduje przyjęcie przez wszystkie bajty pamięci wartości 0FFh. Odczyt dowolnego bajtu pamięci przed zakończeniem operacji wymazywania będzie dawał wartość 00h. Właściwość tę można wykorzystać do określenia momentu zakończenia operacji wymazywania. Na podobnej zasadzie można również określić koniec operacji zapisu nowego bajtu do pamięci – w tym przypadku próby odczytu programowanego bajtu wykonane przed końcem operacji zapisu będą dawały wartość inną niż programowana (zanegowany będzie najmłodszy bit programowanego bajtu).

Tab. 6.7. Wykaz komend szeregowego programowania pamięci Flash mikrokontrolerów 89S51 i 89S52

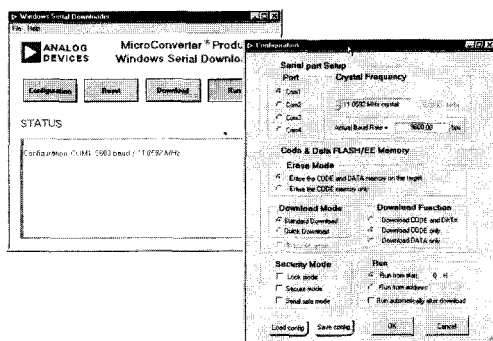
Operacja	Format komendy (4 bajty)	Opis
Komenda rozpoczęcia programowania szeregowego	10101100 01010011 XXXXXXXX XXXXXXXX	Wprowadza mikrokontroler w tryb programowania szeregowego, jeśli wyprowadzenie RST jest w stanie wysokim. Przy poprawnej inicjalizacji programowania szeregowego podczas transmisji czwartego bajtu komendy linią MISO jest zwracany bajt potwierdzenia 01101001.
Wymazywanie pamięci	1010110 0100XXXX XXXXXXXX XXXXXXXX	Kasuje zawartość pamięci programu i pamięci danych typu EEPROM. Czas trwania operacji ok. 16 ms.
Odczyt pamięci programu (tryb bajtowy)	00100000 XXXXXXXX młodszy bajt adresu XXXXXXXX	Odczytuje zawartość wybranego bajtu pamięci programu. Wartość odczytywanego bajtu jest przesyłana linią MISO podczas transmisji czwartego bajtu komendy. AAAAA – 5 starszych bitów adresu (A12...8)
Programowanie pamięci programu (tryb bajtowy)	01000000 XXXXXXXX młodszy bajt adresu bajt do zapisu	Programuje wybrany bajt pamięci programu. AAAAA – 5 starszych bitów adresu (A12...8)
Odczyt pamięci programu (tryb stronicowy)	00110000 XXXXXXXX młodszy bajt adresu XXXXXXXX	Odczytuje zawartość wybranej 256-bajtowej strony pamięci programu. AAAAA – 5 starszych bitów adresu, czyli adres strony (A12...8) Począwszy od trzeciego bajtu komendy, linią MISO przesyłane są kolejne bajty (od 0 do 255) odczytywanej pamięci programu.
Programowanie pamięci programu (tryb stronicowy)	01010000 XXXXXXXX bajt 0 do zapisu bajt 1 do zapisu...	Programuje wybraną 256-bajtową stronę pamięci programu. AAAAA – 5 starszych bitów adresu, czyli adres strony (A12...8) Począwszy od trzeciego bajtu komendy przesyłane są kolejne bajty (od 0 do 255) do zaprogramowania
Programowanie bitów zabezpieczających	10101100 111000NN XXXXXXXX XXXXXXXX	Programuje bity zabezpieczające. NN – binarnie zakodowany numer bitu zabezpieczającego do zaprogramowania; bity zabezpieczające należy programować kolejno
Odczyt bitów zabezpieczających	00101000 XXXXXXXX XXXXXXXX XXXXXXXX	Odczytuje stan bitów zabezpieczających. Wartość bitów zabezpieczających jest przesyłana linią MISO podczas transmisji czwartego bajtu komendy jako XXBBBXX, gdzie BBB – bity zabezpieczające B3...1 (stan wysoki oznacza bit zaprogramowany)

6.2.5. Pamięć Flash w wybranych mikrokontrolerach innych firm

Wprawdzie najbardziej rozpowszechnionymi obecnie w Polsce mikrokontrolerami rodziny '51 z pamięcią programu typu Flash są układy firmy Atmel, warto jednak wspomnieć jeszcze o co najmniej dwóch producentach, którzy oferują bardzo interesujące układy rodziny '51 – są to Analog Devices i Winbond.

Analog Devices jest firmą doskonale znaną przede wszystkim z wyrafinowanych układów analogowych, lecz od pewnego czasu produkuje ona także mikrokontrolery z wysokiej klasy przetwornikami A/C i C/A. Są to dość rozbudowane mikrokontrolery rodziny '51 (stąd też nie zostały one w książce szerzej opisane), wyposażone m.in. w pamięć programu typu Flash, która może być programowana za pomocą standardowego łącza szeregowego. Zawartość tej pamięci może być zatem łatwo modyfikowana także po zamontowaniu mikrokontrolera w urządzeniu (tzw. programowanie wewnętrzukładowe).

Żeby wprowadzić mikrokontroler w tryb programowania szeregowego, należy na czas załączania zasilania lub zerowania zewnętrznym sygnałem RESET wyprowadzenie PSEN dołączyć do masy za pośrednictwem rezystora o wartości 1 k Ω . Po wykryciu takiego stanu, mikrokontroler samoczynnie konfiguruje parametry pracy łącza szeregowego: transmisja asynchroniczna, 8 bitów danych, brak bitu parzystości, prędkość transmisji 9600 bodów (dla $f_{OSC} = 11,0592 \text{ MHz}$), a następnie przechodzi do wykonywania programu ładującego, który umożliwia przeprogramowanie wewnętrznej pamięci Flash przez dowolne urządzenie zewnętrzne. Najwygodniej jest wykorzystać do tego celu komputer klasy PC wyposażony w interfejs RS-232 i system operacyjny Windows, gdyż Analog Devices bezpłatnie udostępnia program *WSD.EXE* (*Windows Serial Downloader* – rys. 6.5), który uwalnia użytkownika od konieczności poznawania protokołu transmisji programu ładującego – do zaprogramowania mikrokontrolera za pomocą *WSD.EXE* wystarczy plik typu Intel HEX z zawartością, jaka ma być umieszczona w programowalnej pamięci Flash oraz znajomość częstotliwości taktowania mikrokontrolera i numeru wykorzystywanego w komputerze portu szeregowego (COM1...COM4).



Rys. 6.5. Widok okien: głównego i konfiguracji programu WSD firmy Analog Devices

Zbliżoną technikę programowania zastosowała firma Winbond, która także produkuje całą serię mikrokontrolerów rodziny '51 z pamięcią programu typu Flash. Podstawową różnicą w stosunku do układów Analog Devices jest fakt, że Winbond nie umieszcza w swoich mikrokontrolerach żadnego programu ładującego, a jedynie dzieli wewnętrzną pamięć programu na dwa obszary, z których jeden (LDROM) może być wykorzystany jako pamięć



Program Windows Serial Downloader jest dostępny w Internecie pod adresem:

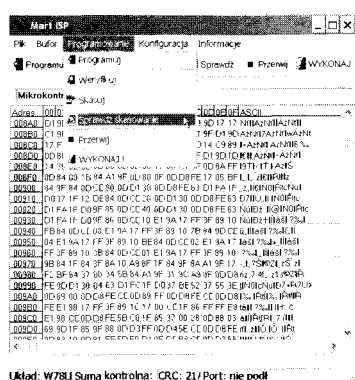
[ftp://ftp.analog.com/pub/www/technology/dataConverters/microconverter/](http://ftp.analog.com/pub/www/technology/dataConverters/microconverter/)



Mikrokontrolery z rodziny '51 firmy Winbond z wbudowaną pamięcią Flash są oferowane na polskim rynku z bootloaderem opracowanym przez firmę Marthel. W związku z tym są one przygotowane do programowania ISP.

programu ładującego, a drugi (APROM) – jako standardowa pamięć wewnętrzna właściwego programu użytkownika. Przejsie do wykonywania *firmware'u* umieszczonego w LDROM następuje po wyjściu mikrokontrolera z uśpienia (tylko po uprzednim ustawieniu odpowiedniej zawartości rejestru CHPCON), względnie wskutek wymuszenia niskiego stanu na wyprowadzeniach P2.6 i P2.7 lub na wyprowadzeniu P4.3 (na co najmniej 1 μ s przed końcem i co najmniej 24 cykle zegarowe po zakończeniu zerowania mikrokontrolera zewnętrznym sygnałem RESET). Podstawową zaletą i wadą zarazem rozwiązania proponowanego przez firmę Winbond jest możliwość, ale i konieczność, wstępnego opracowania i zaprogramowania własnego programu ładującego. Zaletą, gdyż możliwe jest dowolne skonstruowanie programu ładującego – przystosowanie go do programowania mikrokontrolera za pomocą łącza szeregowego, programowo zaimplementowanego interfejsu SPI lub dowolnego innego (także nie-standardowego) sposobu wymiany informacji. Wadą, gdyż użytkownik musi wstępnie zaprogramować mikrokontroler w sposób standardowy (umieszczając w nim program ładujący), by docelowo możliwa była realizacja programowania wewnątrzukładowego za pomocą np. łącza szeregowego. Wadę tę niweluje do pewnego stopnia oferta firmy Marthel (dystrybutora układów Winbonda), która opracowała własny program ładujący (oraz *MartISP* – oprogramowanie współpracujące z nim od strony komputera) i sprzedaje mikrokontrolery Winbonda z wpisanym do pamięci LDROM wspomnianym *firmwarem* bez konieczności ponoszenia dodatkowych kosztów.

Oprogramowanie *MartISP* (rys. 6.6) jest przystosowane do pracy w systemie Windows (95/98/2000/XP) i podobnie jak program ładujący, jest przez firmę Marthel udostępniane bezpłatnie.



Rys. 6.6. Wygląd okna programu *MartISP*

Program *MartISP* obsługuje następujące mikrokontrolery firmy Winbond:

- W78E58B – 32 KB APROM, 4 KB LDROM,
- W78LE58 – 32 KB APROM, 4 KB LDROM,
- W78E858 – 32 KB APROM, 4 KB LDROM,
- W78E516 – 64 KB APROM, 4 KB LDROM,
- W78LE516 – 64 KB APROM, 4 KB LDROM,
- W77E516 – 64 KB APROM, 4 KB LDROM,
- W77E532 – 128 KB APROM, 4 KB LDROM.



Na stronie <http://www.btc.pl/index.php?id=mcs51> są dostępne kody źródłowe procedur, za pomocą których można odczytywać i programować zawartość pamięci Flash w mikrokontrolerach '51 firmy Winbond. Udostępniła je firma Marthel, która jest dystrybutorem firmy Winbond w Polsce.

**Lista instrukcji
mikrokontrolerów
rodziny '51**



Lista instrukcji mikrokontrolerów rodziny '51 zawiera 111 instrukcji, z których 49 jest jednobajtowych, 45 dwubajtowych i 17 trybajtowych. Lista ta jest stała i jednakowa dla większości mikrokontrolerów rodziny '51. Wyjątkiem pod tym względem są m.in. mikrokontrolery o zredukowanej liczbie wyprowadzeń, ponieważ nie umożliwiają one korzystania z zewnętrznej pamięci programu i danych. W mikrokontrolerach tych nie można zatem stosować instrukcji MOVX, a argumenty adresowe instrukcji skoków, wywołań procedur itp. muszą być dobrane tak, by wykonanie tych instrukcji nie powodowało przekroczenia obszaru wewnętrznej pamięci programu.

7.1. Tryby adresowania

Instrukcje mikrokontrolerów rodziny '51 wykorzystują pięć trybów adresowania:

- rejestrowe,
- bezpośrednie,
- natychmiastowe,
- pośrednie zawartością rejestru,
- pośrednie sumą zawartości rejestru bazowego i indeksowego.

W adresowaniu rejestrowym, operandem (argumentem instrukcji) jest jeden z rejestrów R0...7 (oczywiście z aktywnego banku rejestrów), akumulator ACC, rejestr B, wskaźnik danych DPTR lub wskaźnik przeniesienia CY (traktowany umownie jako akumulator dla operacji na pojedynczych bitach).

W adresowaniu bezpośrednim operand jest jawnie podanym, 8-bitowym adresem rejestru SFR, bajtu dolnego obszaru wewnętrznej pamięci RAM lub bitu adresowalnego bezpośrednio (patrz rozdz. 2.1).

W adresowaniu natychmiastowym operand jest stałą (umieszczoną w pamięci programu).

Adresowanie pośrednie zawartością rejestru polega na określaniu adresu za pomocą zawartości rejestrów R0 lub R1 (znajdujących się w aktywnym banku rejestrów). Rejestry te są 8-bitowe, tak więc adresowanie pośrednie wykorzystuje zawartość rejestru jako wskaźnik bajtu znajdującego się w wewnętrznej pamięci RAM lub 256-bajtowym bloku danych pamięci zewnętrznej. Adresowanie pośrednie występuje również przy wykonywaniu rozkazów PUSH i POP umieszczających lub zdejmujących ze stosu

Tab. 7.1. Dostępność poszczególnych obszarów adresowych mikrokontrolerów rodziny '51, w zależności od stosowanego trybu adresowania

Tryb adresowania	Adresowane zasoby pamięci
Rejestrowe	rejestry R0...7 z aktywnego banku rejestrów, akumulator, akumulator pomocniczy (B), wskaźnik CY (tylko dla operacji na pojedynczych bitach), wskaźnik DPTR
Bezpośrednie	dolny obszar wewnętrznej pamięci RAM, rejestry SFR, bity adresowalne bezpośrednio
Natychmiastowe	pamięć programu
Pośrednie zawartością rejestru	wewnętrzna pamięć RAM (adresowanie za pomocą R0, R1, SP), zewnętrzna pamięć danych (adresowanie za pomocą R0, R1, DPTR)
Pośrednie sumą zawartości rejestru bazowego i indeksowego	pamięć programu (adresowanie sumą zawartości DPTR i akumulatora lub licznika rozkazów i akumulatora)

bajt danych. Podczas wykonywania tych rozkazów pamięć (wewnętrzna pamięć RAM) jest adresowana zawartością rejestru SP (wskaźnika stosu). Jedyną formą adresowania pośredniego wykorzystującą wskaźnik 16-bitowy, umożliwiającą adresowanie zewnętrznej pamięci danych w zakresie pełnych 64 KB, są operacje z wykorzystaniem wskaźnika DPTR.

Adresowanie pośrednie sumą zawartości rejestru bazowego i indeksowego umożliwia odczytanie bajtu pamięci programu, którego adres jest określony tą sumą.

7.2. Przegląd instrukcji

Lista instrukcji mikrokontrolerów rodziny '51 zawiera polecenia umożliwiające przesyłanie danych, wykonywanie operacji arytmetycznych i logicznych oraz sterowanie wykonywaniem programu (skoki, wywołania procedur). Na szczególną uwagę zasługuje obecność znacznej liczby instrukcji operujących na pojedynczych bitach, które w przypadku mikrokontrolerów (z racji ich stosowania głównie w układach sterowania) są niezwykle przydatne i bardzo często wykorzystywane. Dużym ułatwieniem z punktu widzenia tworzenia oprogramowania jest występowanie na liście instrukcji mikrokontrolerów rodziny '51 operacji mnożenia i dzielenia. Instrukcje te działają wprawdzie tylko na 8-bitowych liczbach bez znaku, jednakże w przypadku przeciętnych zastosowań mikrokontrolerów są to możliwości wystarczające.

Istotnym efektem wykonywania poszczególnych instrukcji jest modyfikacja wskaźników stanu CPU. Mikrokontrolery rodziny '51 mają cztery takie wskaźniki: P, CY, AC, OV – wszystkie umieszczone w rejestrze PSW. Wskaźnik parzystości P aktualizowany jest po każdej operacji, natomiast wskaźniki CY, AC i OV tylko przez niektóre instrukcje (**tab. 7.2**). W przedstawionym dalej opisie instrukcji mikrokontrolerów rodziny '51 podano szczegółowe zasady zmiany stanu wskaźników. Na szczególną uwagę zasługuje fakt, że wskaźnik przeniesienia nie jest nigdy modyfikowany przez instrukcje inkrementacji.

W zamieszczonym na następnych stronach opisie poszczególnych instrukcji stosowane są następujące oznaczenia:

Rn	-	rejestr R0...7 z aktywnego banku rejestrów ($n = 0...7$),
direct	-	adres rejestru SFR lub bajtu z dolnego obszaru wewnętrznej pamięci RAM,
bit	-	adres bitu w obszarze rejestrów SFR lub wewnętrznej pamięci RAM,
@Ri	-	adresowanie pośrednie zawartością rejestru R0 lub R1 ($i = 0, 1$),
@DPTR	-	adresowanie pośrednie zawartością wskaźnika DPTR,
#data	-	stała 8-bitowa podana jawnie w kodzie instrukcji,
#data16	-	stała 16-bitowa podana jawnie w kodzie instrukcji,
addr16	-	16-bitowy adres skoku (w przypadku instrukcji LJMP) lub początku procedury (w przypadku instrukcji LCALL),
addr11	-	11-bitowy adres skoku (w przypadku instrukcji AJMP) lub początku procedury (w przypadku instrukcji ACALL),
rel	-	liczba 8-bitowa ze znakiem w kodzie uzupełnienia do dwóch, stanowiąca względną wartość skoku SJMP i pozostałych skoków warunkowych.

Ponadto w nawiasach trójkątnych są umieszczone ogólne opisy operandów, nawiasy okrągłe są używane do oznaczenia adresowania bezpośredniego, zaś nawiasy kwadratowe sygnalizują adresowanie pośrednie.

Tab. 7.2. Wykaz instrukcji oddziaływujących na stan wskaźników CY, AC i OV. W tabeli nie uwzględniono instrukcji odwołujących się do rejestru wskaźników stanu (np. MOV PSW, #0), ani instrukcji bitowych działających bezpośrednio na wskaźnikach (np. SETB OV, czy ORL C, bit)

Instrukcja	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RRC	X		
RLC	X		
CJNE	X		

ACALL

<adres_11-bitowy>

Funkcja:

Opis:

Przykład:

Bezwarunkowe wywołanie procedury

ACALL powoduje wywołanie procedury rozpoczynającej się pod podanym adresem bezwzględnym. Wykonanie instrukcji powoduje dwukrotną inkrementację licznika rozkazów, tak by zawierał on adres następnej instrukcji, następnie umieszczenie obu bajtów licznika rozkazów na stosie (najpierw młodszy bajt) i zwiększenie wartości wskaźnika stosu o dwa. Adres wywoływanej procedury jest otrzymywany przez połączenie pięciu najstarszych bitów licznika rozkazów, bitów 7...5 pierwszego bajtu kodu instrukcji i całego drugiego bajtu kodu instrukcji. W związku z tym wywoływana procedura musi się rozpoczynać w obrębie tej samej 2K-bajtowej strony, w której znajduje się pierwszy bajt instrukcji następującej bezpośrednio po wykonywanej instrukcji ACALL. Wykonanie operacji nie zmienia stanu żadnego ze wskaźników.

Początkowy stan wskaźnika SP jest równy 07h. Etykieta SUBR oznacza pamięć programu o adresie 0345h. Po wykonaniu instrukcji

ACALL SUBR

umieszczonej pod adresem 0123h, wskaźnik SP będzie zawierał 09h, bajty wewnętrznej pamięci RAM o adresach 08h i 09h będą zawierały odpowiednio 23h i 01h, zaś licznik rozkazów przyjmie wartość 0345h.

Operacja: **ACALL**
 $PC := PC + 2$
 $SP := SP + 1$
 $[SP] := PC_{7-0}$
 $SP := SP + 1$
 $[SP] := PC_{15-8}$
 $PC_{10-0} := a_{10-0}$ (adres w obszarze bieżącej strony)

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:

a10	a9	a8	1	0	0	0	1
a7	a6	a5	a4	a3	a2	a1	a0

ADD

A, <bajt_źródłowy>

Funkcja: Dodawanie

Opis: ADD powoduje dodanie zawartości wskazanej zmiennej i akumulatora i pozostawia wynik operacji w akumulatorze. Wskaźnik przeniesienia i pomocniczy wskaźnik przeniesienia są ustawiane, jeśli nastąpiło przeniesienie odpowiednio z 7. lub 3. bitu. W przeciwnym razie (brak przeniesień) wskaźniki są zerowane. W przypadku dodawania liczb bez znaku wskaźnik przeniesienia funkcjonuje jako wskaźnik przepełnienia.

Bit OV ustawiany jest tylko wtedy, gdy wystąpiło przeniesienie z bitu 6., przy braku przeniesienia z bitu 7. lub gdy wystąpiło przeniesienie z bitu 7., przy braku przeniesienia z bitu 6. W pozostałych przypadkach OV jest zerowany. Jeśli dodawane są liczby ze znakiem, OV sygnalizuje otrzymanie dodatniego wyniku przy dodawaniu dwóch liczb ujemnych, bądź ujemnego wyniku przy dodawaniu dwóch liczb dodatnich.

Do adresowania operandu źródłowego można wykorzystać jeden z następujących trybów adresowania: rejestrowe, bezpośrednie, pośrednie zawartością rejestru lub natychmiastowe.

Przykład: Akumulator zawiera 0C3h (11000011b), a rejestr R0 zawiera 0AAh (10101010b). Instrukcja

ADD A, R0

pozostawi w akumulatorze wartość 6Dh (01101101b), wyzeruje pomocniczy wskaźnik przeniesienia AC i spowoduje ustawienie wskaźnika przeniesienia CY oraz bitu OV.

ADD A, Rn

Operacja: **ADD**

$A := A + R_n$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

0	0	1	0	1	n	n	n
---	---	---	---	---	---	---	---

ADD A, direct

Operacja: ADD
 $A := A + (\text{direct})$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:	0	0	1	0	0	1	0	1
direct								

ADD A, @Ri

Operacja: ADD
 $A := A + [R_i]$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:	0	0	1	0	0	1	1	i
-----------------	---	---	---	---	---	---	---	---

ADD A, #data

Operacja: ADD
 $A := A + \text{data}$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:	0	0	1	0	0	1	0	0
data								

ADDC

A, <bajt_źródłowy>

Funkcja: Dodawanie z przeniesieniem

Opis: ADCC powoduje dodanie zawartości wskazanej zmiennej, akumulatora oraz bitu przeniesienia i pozostawia wynik operacji w akumulatorze. Wskaźnik przeniesienia i pomocniczy wskaźnik przeniesienia są ustawiane, jeśli nastąpiło przeniesienie odpowiednio z 7. lub 3. bitu. W przeciwnym razie (brak przeniesień) wskaźniki są zerowane. W przypadku dodawania liczb bez znaku wskaźnik przeniesienia funkcjonuje jako wskaźnik przepełnienia. Bit OV ustawiany jest tylko wtedy, gdy wystąpiło przeniesienie z bitu 6., przy braku przeniesienia z bitu 7. lub gdy wystąpiło przeniesienie z bitu 7., przy braku przeniesienia z bitu 6. W pozostałych przypadkach OV jest zerowany. Jeśli dodawane są liczby ze znakiem, OV sygnalizuje otrzymanie dodatniego wyniku przy dodawaniu dwóch liczb ujemnych, bądź ujemnego wyniku przy dodawaniu dwóch liczb dodatnich.

Do adresowania operandu źródłowego można wykorzystać jeden z następujących trybów adresowania: rejestrowe, bezpośrednie, pośrednie zawartością rejestru lub natychmiastowe.

Przykład: Akumulator zawiera 0C3h (11000011b), rejestr R0 zawiera 0AAh (10101010b), a wskaźnik przeniesienia jest ustawiony. Instrukcja

ADDC A, R0

pozostawi w akumulatorze wartość 6Eh (01101110b), wyzeruje pomocniczy wskaźnik przeniesienia AC i spowoduje ustawienie wskaźnika przeniesienia CY oraz bitu OV.

ADDC A, Rn

Operacja: **ADDC**
 $A := A + R_n + CY$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

0	0	1	1	1	n	n	n
---	---	---	---	---	---	---	---

ADDC A, direct

Operacja: **ADDC**
 $A := A + (\text{direct}) + CY$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	0	1	1	0	1	0	1
direct							

ADDC A, @Ri

Operacja: **ADDC**
 $A := A + [R_i] + CY$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

ADDC A, #data

Operacja: **ADDC**
 $A := A + \text{data} + CY$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	0	1	1	0	1	0	0
data							

AJMP

<adres_11-bitowy>

Funkcja: Skok bezwarunkowy

Opis: **AJMP** powoduje przeniesienie wykonywania programu pod podany adres bezwzględny. Wykonanie instrukcji powoduje przyjęcie przez licznik rozkazów wartości otrzymanej przez połączenie pięciu najstarszych bitów licznika rozkazów, bitów 7...5 pierwszego bajtu kodu instrukcji i całego drugiego bajtu kodu instrukcji. W związku z tym miejsce, do którego wykonywany jest skok

Przykład: musi się znaleźć w obrębie tej samej 2K-bajtowej strony, w której znajduje się pierwszy bajt instrukcji następującej bezpośrednio po wykonywanej instrukcji AJMP. Wykonanie operacji nie zmienia stanu żadnego ze wskaźników.
Etykieta JMPADR oznacza pamięć programu o adresie 0345h. Po wykonaniu instrukcji

Operacja: AJMP umieszczona pod adresem 0123h licznik rozkazów przyjmie wartość 0345h.


Liczba cykli: 2
Liczba bajtów: 2
Kod instrukcji: AJMP
PC := PC + 2
PC₁₀₋₀ := a₁₀₋₀ (adres w obszarze bieżącej strony)

a10	a9	a8	0	0	0	0	1
a7	a6	a5	a4	a3	a2	a1	a0

ANL

<bajt_przeznaczenia>, <bajt_źródłowy>

Funkcja: Logiczna funkcja AND dwóch zmiennych
Opis: ANL wykonuje bitowy iloczyn logiczny dwóch wskazanych zmiennych bajtowych (każdy bajt traktowany jest jako zbiór ośmiu bitów). Wynik operacji umieszczany jest w pierwszej z wymienionych zmiennych. Wykonanie operacji nie zmienia stanu żadnego ze wskaźników.
Możliwych jest 6 różnych kombinacji trybów adresowania dla dwóch używanych przez instrukcję operandów. Jeśli miejscem przeznaczenia wyniku jest akumulator, to zmienna źródłowa może być adresowana w trybie rejestrowym, bezpośrednim, pośrednim (zawartością rejestru) lub natychmiastowym. Jeśli miejsce przeznaczenia wyniku jest adresowane bezpośrednio, to zmienna źródłowa może być akumulatorem lub zmienną adresowaną w trybie natychmiastowym.



Jeśli instrukcja jest wykorzystywana do zmiany stanu linii portu, to wartością użytą jako pierwotny stan portu będzie wartość odczytana z rejestru wyjściowego portu, a nie rzeczywisty stan wyprowadzeń portu.

Przykład: Jeśli akumulator zawiera 0C3h (11000011b), rejestr R0 zawiera 0AAh (10101010b), to instrukcja
ANL A, R0
pozostawi w akumulatorze wartość 82h (10000010b).
Instrukcję można wykorzystywać do wyzerowania bitów wybranego bajtu wewnętrznej pamięci RAM lub rejestru, jeśli miejscem

przeznaczenia wyniku jest bajt adresowany bezpośrednio. Maską określającą, które bity będą zerowane jest wówczas stała wyszczególniona w instrukcji lub zawartością akumulatora (zwykle używaną z poprzedzających instrukcję obliczeń). Instrukcja

ANL P1, #01110011B

spowoduje wyzerowanie bitów 7, 3 i 2 portu P1.

ANL A, Rn

Operacja: **ANL**

$A := A \text{ and } R_n$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

0	1	0	1	1	n	n	n
---	---	---	---	---	---	---	---

ANL A, direct

Operacja: **ANL**

$A := A \text{ and (direct)}$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	0	1	0	1	0	1
direct							

ANL A, @Ri

Operacja: **ANL**

$A := A \text{ and } [R_i]$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

ANL A, #data

Operacja: **ANL**

$A := A \text{ and data}$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	0	1	0	1	0	0
data							

ANL direct, A

Operacja: **ANL**

$(\text{direct}) := A \text{ and (direct)}$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	0	1	0	0	1	0
direct							

ANL C, /bit

Operacja: ANL
 $CY := CY \wedge \sim(\text{bit})$

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:

1	0	1	1	0	0	0	0
bit							

CJNE

<bajt_przeznaczenia>, <bajt_źródłowy>, <adres_względny>

Funkcja: Porównanie i skok jeśli różne

Opis: CJNE porównuje wartości dwóch pierwszych operandów i wykonuje skok jeśli są one różne. Wartość adresu, do którego wykonywany jest skok, jest obliczana przez dodanie przesunięcia (ostatniego z operandów będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów tak, by wskazywał on instrukcję znajdującą się za wykonywaną instrukcją CJNE. Jeśli <bajt_przeznaczenia> traktowany jako liczba bez znaku jest mniejszy niż analogicznie interpretowany <bajt_źródłowy>, to wskaźnik przeniesienia jest ustawiany. W przeciwnym razie (odwrotny wynik porównania) bit przeniesienia jest zerowany. Żaden z operandów nie ulega zmianie. Możliwe są cztery różne kombinacje adresowania pierwszych dwóch operandów: operacja może porównywać zawartość akumulatora z dowolnym bajtem adresowanym bezpośrednio lub natychmiastowo, lub przeprowadzać porównanie rejestru roboczego lub pośrednio adresowanego bajtu pamięci RAM z argumentem adresowanym w trybie natychmiastowym (stałą).

Przykład: Akumulator zawiera 34h. Rejestr R7 zawiera 56h. Pierwsza instrukcja w sekwencji

```

        CJNE R7, #60H, NOT_EQ
        ...           ; R7 = 60H
NOT_EQ: JC REQ_LOW   ; skok jeśli R7 < 60H
        ...           ; R7 > 60H

```

ustawi wskaźnik przeniesienia i spowoduje skok do adresu oznaczonego etykietą NOT_EQ. Znajdująca się tam instrukcja, testując stan wskaźnika przeniesienia, określi czy zawartość R7 jest większa, czy mniejsza niż 60h.

Jeśli stan wyprowadzeń portu P1 także stanowi wartość 34h, to instrukcja

```
WAIT: CJNE A, P1, WAIT
```

wyzeruje bit przeniesienia i przejdzie do wykonania kolejnej instrukcji, jako że stan portu P1 i akumulatora są identyczne. Jeśli stan wyprowadzeń portu P1 będzie różny od 34h, to program będzie powtarzał pętlę zawierającą instrukcję z etykietą WAIT tak długo, aż stan wyprowadzeń portu przyjmie wartość 34h.

CJNE @Ri, #data, rel

Operacja: CJNE
 PC := PC + 3
 IF [R_i] < > data THEN
 PC := PC + rel
 IF [R_i] < data THEN
 CY := 1
 ELSE
 CY := 0

Liczba cykli: 2

Liczba bajtów: 3

Kod instrukcji:

1	0	1	1	0	1	1	i
data							
rel							

CLR**A**

Funkcja: Zerowanie akumulatora

Opis: Akumulator jest zerowany – wszystkie bity przyjmują wartość 0. Żaden ze wskaźników nie ulega zmianie.

Przykład: Akumulator zawiera 5Ch (01011100b). Instrukcja

CLR A

spowoduje przyjęcie przez akumulator stanu 00h (00000000b).

Operacja: CLR

A := 0

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

CLR**<bit>**

Funkcja: Zerowanie bitu

Opis: Wskazany bit jest zerowany. Żaden ze wskaźników nie ulega zmianie. Instrukcja może być przeprowadzona wyłącznie na wskaźniku przeniesienia lub dowolnym bicie adresowalnym bezpośrednio.

Przykład: Rejestr portu P1 zawiera 5Dh (01011101b). Instrukcja

CLR P1.2

spowoduje zmianę zawartości rejestru portu na 59h (01011001b).

CLR C

Operacja: CLR

CY := 0

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

CLR bit

Operacja: CLR
(bit) := 0

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:	1	1	0	0	0	0	1	0
	bit							

CPL A

Funkcja: Negacja zawartości akumulatora
Opis: Wszystkie bity akumulatora zmieniają swój stan. Bity zawierające zera są ustawiane, natomiast bity zawierające jedynki są zerowane. Żaden ze wskaźników nie ulega zmianie.

Przykład: Akumulator zawiera 5Ch (01011100b). Instrukcja
CPL A
spowoduje przyjęcie przez akumulator stanu 0A3h (10100011b).

Operacja: CPL
A := A xor 0FFh


Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:	1	1	1	1	0	1	0	0
-----------------	---	---	---	---	---	---	---	---

CPL <bit>

Funkcja: Zanegowanie bitu
Opis: Wskazany bit jest negowany. Żaden ze wskaźników nie ulega zmianie. Instrukcja może być przeprowadzona wyłącznie na wskaźniku przeniesienia lub dowolnym bicie adresowalnym bezpośrednio.



Jeśli instrukcja jest używana do zmiany stanu wyprowadzenia portu, to stan początkowy negowanego bitu jest ustalany na podstawie wartości wpisanej do rejestru wyjściowego portu, a nie stanu wyprowadzenia mikrokontrolera.

Przykład: Rejestr portu P1 zawiera 5Dh (01011101b). Instrukcje
CPL P1.1
CPL P1.2
spowodują zmianę zawartości rejestru portu na 5Bh (01011011b).

CPL C
Operacja: CPL
CY := ~CY

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:	1	0	1	1	0	0	1	1
-----------------	---	---	---	---	---	---	---	---

CPL bit

Operacja: CPL
 $(\text{bit}) := \sim(\text{bit})$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:	1	0	1	1	0	0	1	0
bit								

DA**A**

Funkcja:

Poprawka dziesiętna po dodawaniu

Opis:

DA A koryguje zawartość akumulatora, będącą wynikiem dodawania dwóch zmiennych (każda w postaci upakowanej liczby BCD), dając w wyniku bajt zawierający dwie cyfry dziesiętne. Do dodawania przed wykonaniem poprawki dziesiętnej może być wykorzystana dowolna z instrukcji ADD lub ADDC.

Jeśli wartość zakodowana na czterech młodszych bitach akumulatora jest większa od 9 lub ustawiony jest wskaźnik przeniesienia pomocniczego AC, to do zawartości akumulatora jest dodawana liczba 6. Daje to w wyniku poprawną postać czterech młodszych bitów akumulatora, a ponadto ustawia wskaźnik przeniesienia, jeśli w wyniku dodania 6 przeniesienie wystąpiło (jeśli jednak dodawanie pomocnicze nie wywołało przeniesienia, to stan bitu CY nie ulega zmianie).

Jeśli po wykonaniu opisanej wyżej operacji bit przeniesienia jest ustawiony lub wartość zakodowana na czterech starszych bitach akumulatora przekracza 9, to do starszej połówki akumulatora dodawane jest 6. I analogicznie jak poprzednio, daje to poprawną postać czterech starszych bitów akumulatora, a ponadto ustawia wskaźnik przeniesienia, jeśli w wyniku dodania 6 przeniesienie wystąpiło (jeśli jednak dodawanie pomocnicze nie wywołało przeniesienia, to stan bitu CY nie ulega zmianie). Ostatecznie wskaźnik przeniesienia sygnalizuje wynik powyżej 99 i umożliwia tym samym realizację dodawania dziesiętnego wielokrotnej precyzji. Stan wskaźnika OV nie ulega zmianie.

Wszystkie opisane operacje są wykonywane w ciągu jednego cyklu maszynowego. W praktyce realizacja poprawki dziesiętnej sprowadza się do dodania do akumulatora wartości 00h, 06h, 60h lub 66h, w zależności od początkowego stanu akumulatora i rejestru PSW.



Operacja poprawki dziesiętnej nie wykonuje zwykłego przekształcenia liczby szesnastkowej na postać BCD, nie ma też zastosowania do operacji inkrementacji, ani odejmowania liczb dziesiętnych.

Przykład: Akumulator zawiera wartość 56h (01010110b), co stanowi reprezentację BCD dziesiętnej liczby 56. Rejestr R3 zawiera 67h (01100111b), co stanowi reprezentację BCD dziesiętnej liczby 67. Wskaźnik przeniesienia jest ustawiony. Sekwencja operacji

ADDC A, R3

DA A

spowoduje najpierw wykonanie zwykłego dodawania w uzupełnieniu do dwóch, dającego w akumulatorze wynik 0BEh (10111110b). Wskaźniki przeniesienia i przeniesienia pomocniczego zostaną w rezultacie tego dodawania wyzerowane. Wykonanie wówczas poprawki dziesiętnej zmieni stan akumulatora na 24h (00100100b), reprezentujący dwie młodsze cyfy wyniku dodawania liczb 56, 67 i przeniesienia. Poprawka dziesiętna spowoduje też ustawienie bitu przeniesienia, sygnalizując przepełnienie dziesiętne. Suma liczb 56, 67 i 1 jest równa 124.

Zmienne bajtowe w kodzie BCD mogą być inkrementowane lub dekrementowane przez dodanie odpowiednio liczb 01h lub 99h. Jeśli początkowo akumulator zawiera 30h (reprezentacja BCD liczby 30), to sekwencja instrukcji

ADD A, #99H

DA A

ustawi wskaźnik przeniesienia i pozostawi w akumulatorze liczbę 29h, jako że $30 + 99 = 129$. Ignorując bit przeniesienia uzyskuje się $30 - 1 = 29$.

Operacja: DA

IF ($A_{3-0} > 9$) OR ($AC=1$) THEN

$A_{3-0} := A_{3-0} + 6$

IF ($A_{7-4} > 9$) OR ($CY=1$) THEN

$A_{7-4} := A_{7-4} + 6$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

DEC

<bajt>

Funkcja: Dekrementacja

Opis: Wskazany bajt jest dekrementowany o 1. Jeśli początkową wartością jest 00h, to wynikiem operacji jest 0FFh. Stan żadnego ze wskaźników nie ulega zmianie. Operandem może być bajt adresowany bezpośrednio, pośrednio zawartością rejestru, akumulator lub rejestr roboczy.



Jeśli instrukcja jest wykorzystywana do zmiany stanu portu, to wartością dekrementowaną jest zawartość rejestru wyjściowego portu, a nie stan linii portu.

Przykład: Rejestr R0 zawiera 7Fh (01111111b). Bajty wewnętrznej pamięci RAM o adresach 7Eh i 7Fh zawierają odpowiednio 00h i 40h. Sekwencja instrukcji

DEC @R0

DEC R0

DEC @R0

spowoduje przyjęcie przez rejestr R0 wartości 7Eh i nadanie bajtom wewnętrznej pamięci RAM o adresach 7Eh i 7Fh odpowiednio wartości 0FFh i 3Fh.

DEC A

Operacja: DEC

$$A := A - 1$$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

DEC Rn

Operacja: DEC

$$R_n := R_n - 1$$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

0	0	0	1	1	n	n	n
---	---	---	---	---	---	---	---

DEC direct

Operacja: DEC

$$(\text{direct}) := (\text{direct}) - 1$$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	0	0	1	0	1	0	1
direct							

DEC @Ri

Operacja: DEC

$$[R_i] := [R_i] - 1$$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

DIV

AB

Funkcja:

Dzielenie całkowite

Opis:

DIV AB dzieli 8-bitową liczbę bez znaku umieszczoną w akumulatorze przez 8-bitową liczbę bez znaku znajdującą się w rejestrze B. Po wykonaniu operacji akumulator zawiera iloraz, a rejestr B resztę z dzielenia całkowitego. Wskaźniki CY oraz OV są zerowane.

Wyjątek: Jeśli przed wykonaniem operacji w rejestrze B znajdowała się liczba 00h, to wynik dzielenia (stan rejestrów A i B) jest nieokreślony, natomiast wykonanie instrukcji spowoduje ustawienie wskaźnika OV. Tak czy inaczej wskaźnik przeniesienia będzie zerowany.

Przykład:

Akumulator zawiera 251 (0FBh, czyli 11111011b), a rejestr B zawiera 18 (12h, czyli 00010010b). Instrukcja

DIV AB

spowoduje przyjęcie przez akumulator wartości 13 (0Dh, czyli 00001101b), a przez rejestr B wartości 17 (11h, czyli 00010001b), jako że $251 = (13 \times 18) + 17$. Wskaźniki przeniesienia i przepełnienia zostaną wyzerowane.

Operacja:

DIV

A := A div B

B := A mod B

Liczba cykli:

4

Liczba bajtów:

1

Kod instrukcji:

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

DJNZ

<bajt>, <adres_względny>

Funkcja:	Dekrementacja i skok jeśli nie zero
Opis:	<p>DJNZ zmniejsza zawartość wskazanej zmiennej o jeden i, jeśli wynikiem dekrementacji jest wartość różna od zera, wykonuje skok pod podany adres. Wynikiem dekrementacji wartości 00h jest 0FFh. Stan żadnego ze wskaźników nie ulega zmianie. Wartość adresu, do którego wykonywany jest skok, obliczana jest przez dodanie przesunięcia (drugiego z operandów, będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów tak, by wskazywał on instrukcję znajdującą się za wykonywaną instrukcją DJNZ. Dekrementowaną zmienną może być rejestr roboczy lub dowolny bajt adresowany bezpośrednio.</p> <p>Uwaga: Jeśli instrukcja wykorzystywana jest do zmiany stanu portu, to wartością dekrementowaną jest zawartość rejestru wyjściowego portu, a nie stan linii portu.</p>

INC @Ri

Operacja: INC

$$[R_i] := [R_i] + 1$$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

INC**DPTR**

Funkcja:

Inkrementacja wskaźnika danych

Opis:

Zwiększa 16-bitowy wskaźnik danych o jeden. Wykonywana jest inkrementacja 16-bitowa (modulo 2^{16}) – przepełnienie mniej znaczącego bajtu wskaźnika (DPL) z wartości 0FFh na 00h spowoduje inkrementację bardziej znaczącego bajtu wskaźnika (DPH). Stan żadnego ze wskaźników nie ulega zmianie.

Przykład:

Rejestr DPTR jest jedynym 16-bitowym rejestrem, którego zawartość może być inkrementowana za pomocą pojedynczej instrukcji. Rejestry DPH i DPL zawierają odpowiednio 12h i 0FEh. Ciąg instrukcji

INC DPTR

INC DPTR

INC DPTR

spowoduje przyjęcie przez rejestry DPH i DPL wartości 13h i 01h.

Operacja:

INC

$$DPTR := DPTR + 1$$

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

JB

<bit>, <adres_względny>

Funkcja:

Skok jeśli bit jest jedynką

Opis:

Jeśli wskazany bit jest jedynką, to wykonywany jest skok pod podany adres, w przeciwnym razie następuje przejście do instrukcji znajdującej się bezpośrednio za wykonywaną instrukcją skoku warunkowego. Wartość adresu, do którego wykonywany jest skok, jest obliczana przez dodanie przesunięcia (drugiego z operandów, będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów tak, by wskazywał on instrukcję znajdującą się za wykonywaną instrukcją JB. Wartość testowanego bitu nie ulega zmianie. Stan żadnego ze wskaźników nie ulega zmianie.

Liczba cykli: 2

Liczba bajtów: 3

Kod instrukcji:

0	0	0	1	0	0	0	0
bit							
rel							

JC

<adres_względny>

Funkcja:

Skok, jeśli wskaźnik przeniesienia jest jedynką

Opis:

Jeśli wskaźnik przeniesienia jest jedynką, to wykonywany jest skok pod podany adres, w przeciwnym razie następuje przejście do instrukcji znajdującej się bezpośrednio za wykonywaną instrukcją skoku warunkowego. Wartość adresu, do którego wykonywany jest skok, obliczana jest przez dodanie podanego przesunięcia (będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów o dwa. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład:

Wskaźnik przeniesienia jest wyzerowany. Ciąg instrukcji

JC LABEL1

CPL C

JC LABEL2

spowoduje ustawienie wskaźnika przeniesienia i przejście z wykonywaniem programu do miejsca oznaczonego etykietą LABEL2.

Operacja:

JC

PC := PC + 2

IF CY=1 THEN

PC := PC + rel

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:

0	1	0	0	0	0	0	0
rel							

JMP

@A + DPTR

Funkcja:

Skok pośredni

Opis:

Dodaje 8-bitową stałą bez znaku umieszczoną w akumulatorze do 16-bitowego wskaźnika danych i wpisuje tak otrzymany wynik do licznika rozkazów. Następna instrukcja jest pobierana spod adresu określonego nowym stanem licznika rozkazów. Wykonywane dodawanie jest 16-bitowe – przeniesienie z młodszego bajtu wyniku jest wprowadzane do starszego bajtu. Stan akumulatora, ani wskaźnika danych nie ulega zmianie. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład: W akumulatorze jest umieszczona parzysta liczba z zakresu od 0 do 6. Podany ciąg instrukcji spowoduje przejście do jednej z instrukcji AJMP umieszczonych w tabeli skoków zaczynającej się w miejscu oznaczonym etykietą JMPTAB.

```

MOV DPTR, #JMPTAB
JMP @A+DPTR
JMPTAB: AJMP LABEL0
        AJMP LABEL1
        AJMP LABEL2
        AJMP LABEL3

```

Jeśli przed wykonaniem powyższego ciągu instrukcji akumulator będzie zawierał 04h, to rezultatem będzie przeniesienie wykonywania programu do miejsca oznaczonego etykietą LABEL2. Należy pamiętać, że instrukcja AJMP jest dwubajtowa, a zatem kody kolejnych instrukcji skoku umieszczone są co drugi bajt.

Operacja: JMP
PC := A+DPTR

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

JNB

<bit>, <adres_względny>

Funkcja: Skok, jeśli bit jest zerem

Opis: Jeśli wskazany bit jest zerem, to wykonywany jest skok pod podany adres, w przeciwnym razie następuje przejście do instrukcji znajdującej się bezpośrednio za wykonywaną instrukcją skoku warunkowego. Wartość adresu, do którego wykonywany jest skok, obliczana jest przez dodanie przesunięcia (drugiego z operandów, będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów tak, by wskazywał on instrukcję znajdującą się za wykonywaną instrukcją JNB. Wartość testowanego bitu nie ulega zmianie. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład: Stan wyprowadzeń portu P1 jest reprezentowany liczbą 11001010b. Akumulator zawiera liczbę 56h (01010110b). Ciąg instrukcji

```

JNB P1.3, LABEL1
JNB ACC.3, LABEL2

```

spowoduje przejście z wykonywaniem programu do miejsca oznaczonego etykietą LABEL2.

Operacja: JNB

```

PC := PC + 3
IF (bit)=0 THEN
    PC := PC + rel

```

Liczba cykli: 2

Liczba bajtów: 3

Kod instrukcji:	0	0	1	1	0	0	0	0
	bit							
	rel							

JNC

<adres_względny>

Funkcja: Skok, jeśli wskaźnik przeniesienia jest zerem

Opis: Jeśli wskaźnik przeniesienia jest zerem, to wykonywany jest skok pod podany adres, w przeciwnym razie następuje przejście do instrukcji znajdującej się bezpośrednio za wykonywaną instrukcją skoku warunkowego. Wartość adresu, do którego wykonywany jest skok, obliczana jest przez dodanie podanego przesunięcia (będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów o dwa. Stan żadnego ze wskaźników (w tym wskaźnika przeniesienia) nie ulega zmianie.

Przykład: Wskaźnik przeniesienia jest ustawiony. Ciąg instrukcji

```
JNC LABEL1
CPL C
JNC LABEL2
```

spowoduje wyzerowanie wskaźnika przeniesienia i przejście z wykonywaniem programu do miejsca oznaczonego etykietą LABEL2.

Operacja: JNC

```
PC := PC + 2
IF CY=0 THEN
    PC := PC + rel
```

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:	0	1	0	1	0	0	0	0
	rel							

JNZ

<adres_względny>

Funkcja: Skok, jeśli zawartość akumulatora nie jest zerem

Opis: Jeśli którykolwiek z bitów akumulatora jest jedynką, to wykonywany jest skok pod podany adres, w przeciwnym razie następuje przejście do instrukcji znajdującej się bezpośrednio za wykonywaną instrukcją skoku warunkowego. Wartość adresu, do którego wykonywany jest skok, obliczana jest przez dodanie podanego przesunięcia (będącego 8-bitową liczbą ze znakiem) do zawarto-

Przykład:

Operacja:

Liczba cykli:

Liczba bajtów:

Kod instrukcji:

ści licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów o dwa. Zawartość akumulatora nie ulega zmianie. Stan żadnego ze wskaźników nie ulega zmianie.

Akumulator zawiera 00h. Ciąg instrukcji

```
JNZ LABEL1
INC A
JNZ LABEL2
```

spowoduje umieszczenie w akumulatorze liczby 01h i przejście z wykonywaniem programu do miejsca oznaczonego etykietą LABEL2.

JNZ

```
PC := PC + 2
IF A < > 0 THEN
    PC := PC + rel
```

2

2

0	1	1	1	0	0	0	0
rel							

JZ

<adres_względny>

Funkcja:

Opis:

Przykład:

Operacja:

Skok, jeśli zawartość akumulatora jest zerem

Jeśli wszystkie bity akumulatora są zerami, to wykonywany jest skok pod podany adres, w przeciwnym razie następuje przejście do instrukcji znajdującej się bezpośrednio za wykonywaną instrukcją skoku warunkowego. Wartość adresu, do którego wykonywany jest skok, obliczana jest przez dodanie podanego przesunięcia (będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów o dwa. Zawartość akumulatora nie ulega zmianie. Stan żadnego ze wskaźników nie ulega zmianie.

Akumulator zawiera 01h. Ciąg instrukcji

```
JZ LABEL1
DEC A
JZ LABEL2
```

spowoduje przyjęcie przez akumulator wartości 00h i przejście z wykonywaniem programu do miejsca oznaczonego etykietą LABEL2.

JZ

```
PC := PC + 2
IF A=0 THEN
    PC := PC + rel
```

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:

0	1	1	0	0	0	0	0
rel							

LCALL

<adres_16-bitowy>

Funkcja:

Dalekie wywołanie procedury

Opis:

LCALL wywołuje procedurę znajdującą się pod wskazanym adresem. Instrukcja powoduje zwiększenie zawartości licznika rozkazów o trzy, tak by wskazywał następną instrukcję, a następnie przesyła 16-bitową zawartość licznika rozkazów na stos (najpierw mniej znaczący bajt), zwiększając jednocześnie zawartość wskaźnika stosu o dwa. Po wykonaniu powyższych czynności do licznika rozkazów jest ładowany adres procedury podany w drugim i trzecim bajcie kodu instrukcji, powodując tym samym przejście z wykonywaniem programu do załadowanego adresu. Początek procedury może znaleźć się w dowolnym miejscu 64K-bajтового obszaru pamięci programu. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład:

Początkowo wskaźnik stosu zawiera 07h. Etykieta SUBRTN identyfikuje adres 1234h pamięci programu. Po wykonaniu umieszczonej pod adresem 0123h instrukcji

```
LCALL SUBRTN
```

wskaźnik stosu będzie zawierał 09h, bajty wewnętrznej pamięci RAM o adresach 08h i 09h będą zawierały odpowiednio 26h i 01h, a zawartość licznika rozkazów będzie wynosiła 1234h.

Operacja:

LCALL

PC := PC + 3

SP := SP + 1

[SP] := PC₇₋₀

SP := SP + 1

[SP] := PC₁₅₋₈PC₁₅₋₀ := a₁₅₋₀

Liczba cykli: 2

Liczba bajtów: 3

Kod instrukcji:

0	0	0	1	0	0	1	0
a15	a14	a13	a12	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

LJMP

<adres_16-bitowy>

Funkcja: Długi skok

Opis: LJMP powoduje wykonanie bezwarunkowego skoku pod wskazany adres przez załadowanie do licznika rozkazów drugiego i trzeciego bajtu kodu instrukcji. Adres skoku może znaleźć się w dowolnym miejscu 64K-bajтового obszaru pamięci programu. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład: Etykieta JMPADR identyfikuje adres 1234h pamięci programu. Umieszczona pod adresem 0123h instrukcja

LJMP JMPADR

spowoduje przyjęcie przez licznik rozkazów wartości 1234h.

Operacja: LJMP

$PC := a_{15-0}$

Liczba cykli: 2**Liczba bajtów:** 3**Kod instrukcji:**

0	0	0	0	0	0	1	0
a15	a14	a13	a12	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

MOV

<bajt_przeznaczenia>, <bajt_źródłowy>

Funkcja: Przesłanie zmiennej bajtowej

Opis: Zmienna bajtowa wskazana przez drugi z operandów jest kopiowana w miejsce identyfikowane przez pierwszy z operandów. Stan zmiennej źródłowej nie ulega zmianie. Stan żadnego ze wskaźników nie ulega zmianie. Zmienna źródłowa i miejsce przeznaczenia w instrukcji przesłania zmiennej bajtowej mogą być adresowane łącznie na 15 różnych sposobów. Czyni to opisywaną instrukcję najbardziej elastyczną operacją w całej liście rozkazów mikrokontrolerów rodziny '51.

Przykład: Bajt wewnętrznej pamięci RAM znajdujący się pod adresem 30h zawiera 40h, a pod adresem 40h tej pamięci jest umieszczona wartość 10h. Stan wyprowadzeń portu P1 reprezentuje liczba 11001010b (0CAh). Po wykonaniu ciągu instrukcji

```

MOV R0, #30H      ; R0 := 30H
MOV A, @R0        ; A := 40H
MOV R1, A         ; R1 := 40H
MOV B, @R1        ; B := 10H
MOV @R1, P1       ; RAM(40H) := 0CAH
MOV P2, P1        ; P2 := 0CAH

```

w rejestrze roboczym R0 znajdzie się wartość 30h, w rejestrze roboczym R1 i akumulatorze będzie 40h, rejestr B będzie zawierał 10h, a w wewnętrznej pamięci RAM o adresie 40h oraz w rejestrze wyjściowym portu P2 pojawi się 0CAh (11001010b).

MOV A, Rn

Operacja: MOV
 $A := R_n$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

1	1	1	0	1	n	n	n
---	---	---	---	---	---	---	---

MOV A, direct

Operacja: MOV
 $A := (\text{direct})$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

1	1	1	0	0	1	0	1
direct							

MOV A, @Ri

Operacja: MOV
 $A := [R_i]$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

MOV A, #data

Operacja: MOV
 $A := \text{data}$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

1	1	1	0	0	1	0	0
data							

MOV Rn, A

Operacja: MOV
 $R_n := A$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

1	1	1	1	1	n	n	n
---	---	---	---	---	---	---	---

MOV Rn, direct

Operacja: MOV
 $R_n := (\text{direct})$

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:

1	0	1	0	1	n	n	n
direct							

MOV direct, #data

Operacja: MOV
(direct) := data

Liczba cykli: 2

Liczba bajtów: 3

Kod instrukcji:	0	1	1	1	0	1	0	1
direct								
data								

MOV @Ri, A

Operacja: MOV
[R_i] := A

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:	1	1	1	1	0	1	1	i
-----------------	---	---	---	---	---	---	---	---

MOV @Ri, direct

Operacja: MOV
[R_i] := direct

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:	1	0	1	0	0	1	1	i
direct								

MOV @Ri, #data

Operacja: MOV
[R_i] := data

Liczba cykli: 2

Liczba bajtów: 2

Kod instrukcji:	0	1	1	1	0	1	1	i
data								

MOV

<bit_przeznaczenia>, <bit_źródłowy>

Funkcja: Przesłanie zmiennej bitowej

Opis: Zmienna bitowa wskazana przez drugi z operandów jest kopiowana w miejsce identyfikowane przez pierwszy z operandów. Jednym z operandów musi być wskaźnik przeniesienia, drugim może być dowolny bit adresowalny bezpośrednio. Stan żadnego z rejestrów ani bitów, poza bitem przeznaczenia nie ulega zmianie.

Przykład: Początkowo wskaźnik przeniesienia jest ustawiony. Stan wyprowadzeń portu P3 reprezentuje liczba 11000101b, a w rejestrze wyjściowym portu P1 umieszczona jest wartość 35h (00110101b). Wykonanie ciągu instrukcji

```
MOV P1.3, C
MOV C, P3.3
MOV P1.2, C
```

spowoduje wyzerowanie wskaźnika przeniesienia i zmianę zawartości rejestru wyjściowego portu P1 na 39h (00111001b).

MOV C, bit

Operacja: MOV
CY := (bit)

Liczba cykli: 1
Liczba bajtów: 2

Kod instrukcji:	1	0	1	0	0	0	1	0
bit								

MOV bit, C

Operacja: MOV
(bit) := CY

Liczba cykli: 2
Liczba bajtów: 2

Kod instrukcji:	1	0	0	1	0	0	1	0
bit								

MOV

DPTR, #<stała_16-bitowa>

Funkcja: Ładowanie wskaźnika danych stałą 16-bitową
Opis: Wskaźnik danych jest ładowany podaną stałą 16-bitową. Wartość stałej określana jest stanem drugiego (DPH) i trzeciego (DPL) bajtu kodu instrukcji. Stan żadnego ze wskaźników nie ulega zmianie.

Opisywana instrukcja jest jedyną instrukcją przesyłania wartości (stałej lub zmiennej) 16-bitowej.

Przykład: Instrukcja
MOV DPTR, #1234H
spowoduje załadowanie do wskaźnika danych stałej 1234h. Rejestr DPH będzie zawierał 12h, a rejestr DPL 34h.

Operacja: MOV
DPTR := d₁₅₋₀ <=> (DPH := d₁₅₋₈; DPL := d₇₋₀)

Liczba cykli: 2
Liczba bajtów: 3

Kod instrukcji:	1	0	0	1	0	0	0	0
	d15	d14	d13	d12	d11	d10	d9	d8
	d7	d6	d5	d4	d3	d2	d1	d0

MOVC

A, @A + <rejestr_bazowy>

Funkcja: Przesłanie bajtu kodu

Opis: Instrukcja MOVC przesyła do akumulatora bajt kodu lub stałą umieszczoną w pamięci programu. Adres pobieranego bajtu kodu jest określany jako suma zawartości akumulatora (traktowanej jako 8-bitowa liczba bez znaku) i 16-bitowego rejestru bazowego, którym może być wskaźnik danych lub licznik rozkazów. W tym ostatnim przypadku zawartość licznika rozkazów jest wstępnie inkrementowana, tak by przed wykonaniem sumowania z zawartością akumulatora licznik rozkazów wskazywał instrukcję znajdującą się bezpośrednio za wykonywaną instrukcją MOVC. Jeśli rejestrem bazowym jest wskaźnik danych, jego zawartość nie jest zmieniana. Sumowanie wykonywane jest jako 16-bitowe, tak że przeniesienie z młodszego bajtu jest uwzględniane w starszym bajcie. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład: W akumulatorze znajduje się liczba z zakresu od 0 do 3. Poniższy ciąg instrukcji spowoduje umieszczenie w akumulatorze zawartości jednego z bajtów kodu programu, zdefiniowanych za pomocą pseudoinstrukcji DB (definiuj bajtami).

```
REL_PC:  INC A
          MOVC A, @A + PC
          RET
          DB 66H
          DB 77H
          DB 88H
          DB 99H
```

Jeśli procedura REL_PC zostanie wywołana przy zawartości akumulatora równej 01h, to po powrocie z procedury w akumulatorze będzie wartość 77h. Instrukcja INC A umieszczona przed instrukcją MOVC powoduje „przeskoczenie” instrukcji RET, oddzielającej instrukcję MOVC od tablicy stałych. Gdyby tablica była oddzielona od instrukcji MOVC większą liczbą bajtów kodu, to zawartość akumulatora zwiększono by o odpowiednią wartość wykorzystując instrukcję dodawania zamiast instrukcji inkrementacji.

MOVC A, @A+DPTR

Operacja: MOVC
 $A := (A + DPTR)$

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

MOVC A, @A+PC

Operacja: MOVC
 $PC := PC + 1$
 $A := (A + PC)$

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

MOVX

<bajt_przeznaczenia>, <bajt_źródłowy>

Funkcja:

Przesłanie z udziałem zewnętrznej pamięci danych

Opis:

Instrukcja MOVX służy do przeprowadzania wymiany informacji między akumulatorem i zewnętrzną pamięcią danych. Przesłania za pomocą instrukcji MOVX mogą być wykonywane przy wykorzystaniu 8- lub 16-bitowego adresowania zewnętrznej pamięci RAM. Przy adresowaniu 8-bitowym zewnętrzna pamięć może być adresowana zawartością rejestru R0 lub R1. Adres jest wystawiany na porcie P0 i jest multipleksowany z bajtem danych. Adresowanie 8-bitowe jest wystarczające zwykle tylko przy korzystaniu z niewielkiej zewnętrznej pamięci danych lub przy sterowaniu zewnętrznymi układami peryferyjnymi. Jeśli obszar adresowy układów zewnętrznych przekracza 256 bajtów, to do określania starszych bitów adresu przy adresowaniu 8-bitowym mogą być wykorzystane dowolne wyprowadzenia mikrokontrolera (bardzo często wykorzystuje się do tego celu linie portu P2). Stan tych wyprowadzeń musi być oczywiście odpowiednio ustawiany przed każdą operacją MOVX działającą w trybie 8-bitowym.

W przypadku adresowania 16-bitowego do adresowania jest wykorzystywany 16-bitowy wskaźnik danych DPTR. Podczas wykonywania instrukcji starszy bajt adresu (zawartość DPH) wystawiany jest na liniach portu P2, zaś młodszy bajt adresu (DPL) jest multipleksowany z bajtem danych na liniach portu P0. Po wykonaniu operacji przesłania linie portu P2 powracają do stanu sprzed jej wykonywania. Adresowanie 16-bitowe jest szybsze i wydajniejsze niż 8-bitowe w przypadku dużego obszaru adresowego układów zewnętrznych, ponieważ nie jest wtedy potrzebne wstępne ustalanie (za pomocą dodatkowej instrukcji) stanu starszych bitów adresowych przed każdą operacją MOVX.

Możliwe jest naprzemienne stosowanie obu trybów adresowania.

Przykład:

Zewnętrzna pamięć RAM o pojemności 256 bajtów z multipleksowaną szyną danych i adresów (np. układ 8155) jest podłączona do portu P0 mikrokontrolera. Liniami sterującymi są odpowiednie linie portu P3. Linie portów P1 i P2 są wykorzystywane jako zwykle wejścia/wyjścia. Rejestry robocze R0 i R1 zawierają 12h i 34h. Pod adresem 34h zewnętrznej pamięci RAM umieszczono liczbę 56h. Ciąg instrukcji

```
MOVX A, @R1
```

```
MOVX @R0, A
```

spowoduje skopiowanie liczby 56h do akumulatora oraz pod adres 12h zewnętrznej pamięci RAM.

MOVX A, @Ri

Operacja: MOVX
A := [R_i]

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

MOVX A, @DPTR

Operacja: MOVX
A := [DPTR]

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

MOVX @Ri, A

Operacja: MOVX
[R_i] := A

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

MOVX @DPTR, A

Operacja: MOVX
[DPTR] := A

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

MUL

AB

Funkcja: Mnożenie

Opis: MUL AB mnoży dwie 8-bitowe liczby bez znaku umieszczone w akumulatorze i rejestrze B. Mniej znaczący bajt 16-bitowego iloczynu jest umieszczany w akumulatorze, bardziej znaczący bajt w rejestrze B. Jeśli wynik jest większy od 255 (0FFh), to ustawiany jest wskaźnik przepełnienia, jeśli zaś starszy bajt wyniku jest zerem, wskaźnik OV jest zerowany. Wskaźnik przeniesienia jest zawsze zerowany.

Przykład: Początkowo akumulator zawiera liczbę 80 (50h), a rejestr B liczbę 160 (0A0h). Wykonanie instrukcji

MUL AB

da wynik 12800 (3200h), a zatem w rejestrze B znajdzie się liczba 32h, a akumulator zostanie wyzerowany. Wskaźnik przepełnienia będzie ustawiony, a wskaźnik przeniesienia wyzerowany.

Operacja: MUL

$A := (A \times B) \bmod 256$

$B := (A \times B) \div 256$

Liczba cykli: 4

Liczba bajtów: 1

Kod instrukcji:

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

NOP

Funkcja: Wykonanie operacji pustej (wytracanie czasu)

Opis: Następuje przejście do następnej instrukcji. Stan żadnego ze wskaźników ani rejestrów, poza licznikiem rozkazów, nie ulega zmianie.

Przykład: Na wyprowadzeniu P2.7 mikrokontrolera należy wygenerować ujemny impuls trwający 5 cykli maszynowych. Zwykła sekwencja instrukcji CLR/SETB spowodowałaby wygenerowanie impulsu trwającego zaledwie jeden cykl maszynowy, należało by zatem wstawić dodatkowe cztery cykle. Żądany impuls można zatem wygenerować (przy założeniu, że wszystkie przerwania są zablokowane) np. za pomocą ciągu instrukcji

CLR P2.7

NOP

NOP

NOP

NOP

SETB P2.7

Operacja: NOP

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

ORL

<bajt_przeznaczenia>, <bajt_źródłowy>

Funkcja: Logiczna funkcja OR dwóch zmiennych

Opis: ORL oblicza bitową sumę logiczną dwóch wskazanych zmiennych bajtowych (każdy bajt traktowany jest jako zbiór ośmiu bitów). Wynik operacji jest umieszczany w pierwszej z wymienionych zmiennych. Wykonanie operacji nie zmienia stanu żadnego ze wskaźników.

Możliwych jest 6 różnych kombinacji trybów adresowania dla dwóch używanych przez instrukcję operandów. Jeśli miejscem przeznaczenia wyniku jest akumulator, to zmienna źródłowa może być adresowana w trybie rejestrowym, bezpośrednim, pośrednim (zawartością rejestru) lub natychmiastowym. Jeśli miejsce przeznaczenia wyniku jest adresowane bezpośrednio, to zmienna

źródłowa może być akumulatorem lub zmienną adresowaną w trybie natychmiastowym.



Jeśli instrukcja jest wykorzystywana do zmiany stanu linii portu, to wartością użytą jako pierwotny stan portu będzie wartość odczytana z rejestru wyjściowego portu, a nie rzeczywisty stan wyprowadzeń portu.

Przykład:

Jeśli akumulator zawiera 0C3h (11000011b), rejestr R0 zawiera 55h (01010101b), to instrukcja

ORL A, R0

pozostawi w akumulatorze wartość 0D7h (11010111b).

Instrukcję można wykorzystywać do ustawiania bitów wybranego bajtu wewnętrznej pamięci RAM lub rejestru – jeśli miejscem przeznaczenia wyniku jest bajt adresowany bezpośrednio. Maską określającą, które bity będą ustawiane jest wówczas stała wyszczególniona w instrukcji lub zawartością akumulatora (zwykle używaną z poprzedzających instrukcję obliczeń). Instrukcja

ORL P1, #00110010B

spowoduje ustawienie bitów 5, 4 i 1 portu P1.

ORL A, Rn

Operacja:

ORL

$A := A \text{ or } R_n$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

0	1	0	0	1	n	n	n
---	---	---	---	---	---	---	---

ORL A, direct

Operacja:

ORL

$A := A \text{ or } (\text{direct})$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	0	0	0	1	0	1
direct							

ORL A, @Ri

Operacja:

ORL

$A := A \text{ or } [R_i]$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

ORL A, #data

Operacja:

ORL

$A := A \text{ or } \text{data}$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	0	0	0	1	0	0
data							

ORL direct, A

Operacja: ORL

(direct) := A or (direct)

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	0	0	0	0	1	0
direct							

ORL direct, #data

Operacja: ORL

(direct) := (direct) or data

Liczba cykli: 2

Liczba bajtów: 3

Kod instrukcji:

0	1	0	0	0	0	1	1
direct							
data							

ORL

C, <bit_źródłowy>

Funkcja:

Logiczna funkcja OR zmiennych bitowych

Opis:

Instrukcja ustawia wskaźnik przeniesienia, jeśli wartość bitu będącego drugim z podanych operandów jest jedynką. W przeciwnym razie stan wskaźnika przeniesienia pozostaje bez zmian. Operacja nie zmienia stanu pozostałych wskaźników. Jeśli drugi z operandów jest poprzedzony kreską ukośną „/”, to do wykonania operacji jest używana zanegowana wartość wskazanego bitu (stan bitu źródłowego nie ulega jednak zmianie).

Operandami źródłowymi mogą być tylko bity adresowalne bezpośrednio.

Przykład:

Wykonanie ciągu instrukcji

```
MOV C, P1.0 ; przesłanie stanu wypro-
              ; wadzenia portu do wskaź-
              ; nika przeniesienia
ORL C, ACC.7 ; suma logiczna wskaźnika
              ; przeniesienia z bitem
              ; 7 akumulatora
ORL C, /OV   ; suma logiczna wskaźnika
              ; przeniesienia z negacją
              ; wskaźnika OV
```

spowoduje ustawienie wskaźnika przeniesienia tylko wtedy, gdy P1.0=1, ACC.7=1 lub OV=0.

PUSH

<bajt_adresowany_bezpośrednio>

Funkcja:	Umieszczenie bajtu na stosie								
Opis:	Zawartość wskaźnika stosu jest zwiększana o jeden. Następnie zawartość bajtu identyfikowanego przez operand instrukcji jest umieszczana w wewnętrznej pamięci RAM, pod adresem określonym przez zawartość wskaźnika stosu. Stan żadnego z pozostałych rejestrów, ani wskaźników nie ulega zmianie.								
Przykład:	Wskaźnik stosu zawiera liczbę 09h, a wskaźnik danych zawiera 1234h. Ciąg instrukcji PUSH DPH PUSH DPL spowoduje umieszczenie w wewnętrznej pamięci RAM o adresach 0Ah i 0Bh liczb 34h i 12h oraz zwiększenie zawartości wskaźnika stosu do 0Bh.								
Operacja:	PUSH SP := SP + 1 [SP] := (direct)								
Liczba cykli:	2								
Liczba bajtów:	2								
Kod instrukcji:	<table border="1"><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0		

RET

Funkcja:	Powrót z procedury
Opis:	RET pobiera ze stosu dwa bajty, umieszczając je w bardziej i mniej znaczącym bajcie licznika rozkazów oraz zmniejszając zawartość wskaźnika stosu o dwa. Działanie programu jest kontynuowane od ustawionego w ten sposób nowego adresu – z reguły jest to adres instrukcji położonej bezpośrednio za instrukcją ACALL lub LCALL, która wywołała procedurę. Stan żadnego ze wskaźników nie ulega zmianie.
Przykład:	Wskaźnik stosu zawiera liczbę 0Bh, a wewnętrzna pamięć RAM o adresach 0Ah i 0Bh zawiera odpowiednio 23h i 01h. Instrukcja <div style="text-align: center;">RET</div> spowoduje kontynuację programu począwszy od adresu 0123h oraz przyjęcie przez wskaźnik stosu wartości 09h.
Operacja:	RET $PC_{15-8} := [SP]$ $SP := SP - 1$ $PC_{7-0} := [SP]$ $SP := SP - 1$

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

RETI

Funkcja: Powrót z procedury obsługi przerwania

Opis: RETI pobiera ze stosu dwa bajty, umieszczając je w bardziej i mniej znaczącym bajcie licznika rozkazów oraz zmniejsza zawartość wskaźnika stosu o dwa. Przywracana jest możliwość obsługi przerwania o priorytecie takim samym jak przed wystąpieniem przerwania, którego obsługa jest właśnie kończona. Stan żadnego ze wskaźników nie ulega zmianie. Stan rejestru PSW sprzed przerwania nie jest automatycznie odtwarzany. Działanie programu kontynuowane jest z reguły od adresu instrukcji położonej bezpośrednio za instrukcją, podczas wykonywania której wykryte zostało żądanie obsługi przerwania. Instrukcja, od której program będzie kontynuowany, zostanie wykonana przed przejściem do obsługi nowego przerwania nawet wtedy, gdy żądanie nowego przerwania wystąpi już podczas wykonywania instrukcji RETI.

Przykład: Wskaźnik stosu zawiera liczbę 0Bh, a wewnętrzna pamięć RAM o adresach 0Ah i 0Bh zawiera odpowiednio 23h i 01h. Obsługiwane przerwanie zostało wykryte podczas wykonywania instrukcji, której ostatni bajt kodu znajduje się pod adresem 0122h. Instrukcja

RETI

spowoduje kontynuację programu począwszy od adresu 0123h oraz przyjęcie przez wskaźnik stosu wartości 09h.

Operacja: RETI

 $PC_{15-8} := [SP]$ $SP := SP - 1$ $PC_{7-0} := [SP]$ $SP := SP - 1$

Liczba cykli: 2

Liczba bajtów: 1

Kod instrukcji:

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

RL

A

Funkcja: Cykliczne przesunięcie zawartości akumulatora w lewo

Opis: Osiem bitów akumulatora jest przesuwanych w lewo o jeden bit. Przesunięcie jest cykliczne - w wyniku przesunięcia bit 7 akumulatora jest umieszczany na pozycji bitu 0. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład:	Akumulator zawiera liczbę 0C5h (11000101b). Instrukcja RR A spowoduje przyjęcie przez akumulator wartości 0E2h (11100010b). Stan wskaźnika przeniesienia nie ulegnie zmianie.						
Operacja:	RR FOR n := 0 TO 6 DO $A_n := A_{n+1}$ $A_7 := A_0(\text{stare})$						
Liczba cykli:	1						
Liczba bajtów:	1						
Kod instrukcji:	0	0	0	0	0	0	1 1

RRC

A

Funkcja:	Cykliczne przesunięcie zawartości akumulatora w prawo z udziałem wskaźnika przeniesienia						
Opis:	Osiem bitów akumulatora oraz bit CY jest przesuwanych w prawo o jeden bit z przejściem przez wskaźnik CY - bit 0 akumulatora jest przesuwany do wskaźnika przeniesienia. Przesunięcie jest cykliczne - w wyniku przesunięcia początkowa zawartość wskaźnika przeniesienia jest umieszczana na pozycji bitu 7 akumulatora. Stan żadnego ze pozostałych wskaźników nie ulega zmianie.						
Przykład:	Akumulator zawiera liczbę 0C5h (11000101b), a wskaźnik przeniesienia jest wyzerowany. Instrukcja RRC A spowoduje przyjęcie przez akumulator wartości 62h (01100010b). Wskaźnik przeniesienia zostanie ustawiony.						
Operacja:	RRC CY := A_0 FOR n := 0 TO 6 DO $A_n := A_{n+1}$ $A_7 := CY(\text{stare})$						
Liczba cykli:	1						
Liczba bajtów:	1						
Kod instrukcji:	0	0	0	1	0	0	1 1

SETB

<bit>

Funkcja:	Ustawienie bitu
Opis:	Wskazany bit jest ustawiany. Żaden ze wskaźników nie ulega zmianie. Instrukcja może być przeprowadzona wyłącznie na wskaźniku przeniesienia lub dowolnym bicie adresowalnym bezpośrednio.

Przykład: Wskaźnik przeniesienia jest wyzerowany. Rejestr portu P1 zawiera 34h (00110100B). Instrukcje

SETB C

SETB P1.0

spowodują ustawienie bitu przeniesienia i zmianę zawartości rejestru portu P1 na 35h (00110101b).

SETB C

Operacja: SETB

CY := 1

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:	1	1	0	1	0	0	1	1
-----------------	---	---	---	---	---	---	---	---

SETB bit

Operacja: SETB

(bit) := 1

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:	1	1	0	1	0	0	1	0
	bit							

SJMP <adres_względny>

Funkcja: Skok krótki


Opis: Wykonywany jest bezwarunkowy skok pod podany adres. Wartość adresu, do którego wykonywany jest skok, obliczana jest przez dodanie przesunięcia (operandu będącego 8-bitową liczbą ze znakiem) do zawartości licznika rozkazów, po uprzednim zwiększeniu zawartości licznika rozkazów o dwa. Zasięg skoku rozciąga się zatem od 128 bajtów przed do 127 bajtów za wykonywaną instrukcją SJMP. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład: Etykieta RELADR identyfikuje adres 0123h pamięci programu.

Instrukcja

SJMP RELADR

znalazła się pod adresem 0100h. Po jej wykonaniu licznik rozkazów będzie zawierał liczbę 0123h.



UWAGA

W omawianym przykładzie instrukcja występująca bezpośrednio za instrukcją skoku będzie położona pod adresem 0102h. Przesunięcie będzie zatem wynosiło (0123h - 0102h) = 21h. Warto zauważyć, że użycie instrukcji skoku krótkiego, w której wartość adresu względnego będzie równa 0FEh, utworzy nieskończoną pętlę, złożoną z pojedynczej instrukcji.

Operacja: **SJMP** $PC := PC + 2$ $PC := PC + rel$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

1	0	0	0	0	0	0	0
rel							

SUBB

A, <bajt_źródłowy>

Funkcja:

Odejmowanie z pożyczką

Opis:

SUBB powoduje odjęcie zawartości wskazanej zmiennej oraz bitu przeniesienia od zawartości akumulatora i pozostawia wynik operacji w akumulatorze. Wskaźnik przeniesienia (funkcjonujący w tym przypadku jako wskaźnik pożyczki) jest ustawiany, jeśli występuje pożyczka z 8. bitu. W przeciwnym razie (brak pożyczki) wskaźnik przeniesienia jest zerowany. Sygnalizowanie pożyczki umożliwia realizację odejmowania wielokrotnej precyzji. W takim wypadku należy pamiętać, by odejmowanie najmłodszych części liczb było wykonywane przy wyzerowanym wskaźniku przeniesienia (pożyczki). Pomocniczy wskaźnik przeniesienia jest ustawiany, jeśli występuje pożyczka z 4. bitu. W przeciwnym razie pomocniczy wskaźnik przeniesienia jest zerowany. Bit OV jest ustawiany tylko wtedy, gdy wystąpiła pożyczka na bicie 7., przy braku pożyczki na bicie 8. lub gdy wystąpiła pożyczka na bicie 8., przy braku pożyczki na bicie 7. W pozostałych przypadkach OV jest zerowany. Jeśli odejmowane są liczby ze znakiem, OV sygnalizuje otrzymanie dodatniego wyniku przy odejmowaniu liczby dodatniej od liczby ujemnej, bądź ujemnego wyniku przy odejmowaniu liczby ujemnej od liczby dodatniej.

Do adresowania operandu źródłowego można wykorzystać jeden z następujących trybów adresowania: rejestrowe, bezpośrednie, pośrednie zawartością rejestru lub natychmiastowe.

Przykład:

Akumulator zawiera 0C9h (11001001b), rejestr R2 zawiera 54h (01010100b), a wskaźnik przeniesienia jest ustawiony. Instrukcja

SUBB A, R2

pozostawi w akumulatorze wartość 74h (01110100b), wyzeruje pomocniczy wskaźnik przeniesienia AC i wskaźnik przeniesienia CY oraz ustawi bit OV.

Należy zauważyć, że 0C9h minus 54h jest równe 75h. Różnica między tą wartością, a wynikiem otrzymanym w przedstawionym przykładzie wynika z wykonania operacji odejmowania przy ustawionym bicie przeniesienia (pożyczki).

SUBB A, Rn

Operacja: SUBB

$$A := A - R_n - CY$$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

1	0	0	1	1	n	n	n
---	---	---	---	---	---	---	---

SUBB A, direct

Operacja: SUBB

$$A := A - (\text{direct}) - CY$$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

1	0	0	1	0	1	0	1
direct							

SUBB A, @Ri

Operacja: SUBB

$$A := A - [R_i] - CY$$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

SUBB A, #data

Operacja: SUBB

$$A := A - \text{data} - CY$$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

1	0	0	1	0	1	0	0
data							

SWAP**A**

Funkcja:

Zamiana miejscami zawartości połówek akumulatora

Opis:

SWAP A zamienia miejscami zawartość bardziej i mniej znaczącej połówki akumulatora (składających się z 4 bitów każda). Operacja może być rozpatrywana jako cykliczne przesunięcie zawartości akumulatora o cztery bity. Stan żadnego ze wskaźników nie ulega zmianie.

Przykład:

Akumulator zawiera liczbę 0C5h (11000101b). Instrukcja

SWAP A

powoduje przyjęcie przez akumulator wartości 5Ch (01011100b).

Operacja:

SWAP

$$A_{3:0} := A_{7:4}$$

$$A_{7:4} := A_{3:0}(\text{stare})$$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

XCH**A, <bajt>**

Funkcja: Wymiana zawartości akumulatora i zmiennej bajtowej

Opis: XCH powoduje wpisanie do akumulatora zawartości wskazanej zmiennej z jednoczesnym przepisaniem początkowej zawartości akumulatora do tej zmiennej. Drugi z operandów może być rejestrem roboczym, zmienną adresowaną bezpośrednio lub pośrednio zawartością rejestru.

Przykład: Rejestr R0 zawiera liczbę 20h, akumulator 3Fh (00111111b), a bajt wewnętrznej pamięci RAM o adresie 20h zawiera 75h (01110101b). Instrukcja

XCH A, @R0

spowoduje przyjęcie przez bajt wewnętrznej pamięci RAM o adresie 20h wartości 3Fh (00111111b), a przez akumulator wartości 75h (01110101b).

XCH A, Rn

Operacja: XCH

A := R_nR_n := A(stare)

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

1	1	0	0	1	n	n	n
---	---	---	---	---	---	---	---

XCH A, direct

Operacja: XCH

A := (direct)

(direct) := A(stare)

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

1	1	0	0	0	1	0	1
direct							

XCH A, @Ri

Operacja: XCH

A := [R_i][R_i] := A(stare)

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

XCHD

A, @Ri

Funkcja:	Wymiana cyfr								
Opis:	XCHD powoduje wymianę mniej znaczących połówek (bitów 0...3, stanowiących zwykle cyfrę szesnastkową lub dziesiętną w kodzie BCD) akumulatora i bajtu wewnętrznej pamięci RAM (zaadresowanego pośrednio zawartością wskazanego rejestru). Stan żadnego ze wskaźników nie ulega zmianie.								
Przykład:	Rejestr R0 zawiera liczbę 20h, akumulator 36h (00110110b), a bajt wewnętrznej pamięci RAM o adresie 20h zawiera 75h (01110101b). Instrukcja XCHD A, @R0 spowoduje przyjęcie przez bajt wewnętrznej pamięci RAM o adresie 20h wartości 76h (01110110b), a przez akumulator wartości 35h (00110101b).								
Operacja:	XCHD $A_{3-0} := [R_i]_{3-0}$ $[R_i]_{3-0} := A_{3-0}(\text{stare})$								
Liczba cykli:	1								
Liczba bajtów:	1								
Kod instrukcji:	<table><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>i</td></tr></table>	1	1	0	1	0	1	1	i
1	1	0	1	0	1	1	i		

XRL

<bajt_przeznaczenia>, <bajt_źródłowy>

Funkcja:	Logiczna funkcja EXOR dwóch zmiennych
Opis:	XRL oblicza bitową sumę modulo dwa dwóch wskazanych zmiennych bajtowych (każdy bajt traktowany jest jako zbiór ośmiu bitów). Wynik operacji jest umieszczany w pierwszej z wymienionych zmiennych. Wykonanie operacji nie zmienia stanu żadnego ze wskaźników. Możliwych jest 6 różnych kombinacji trybów adresowania dla dwóch używanych przez instrukcję operandów. Jeśli miejscem przeznaczenia wyniku jest akumulator, to zmienna źródłowa może być adresowana w trybie rejestrowym, bezpośrednim, pośrednim (zawartością rejestru) lub natychmiastowym. Jeśli miejsce przeznaczenia wyniku jest adresowane bezpośrednio, to zmienna źródłowa może być akumulatorem lub zmienną adresowaną w trybie natychmiastowym.
Przykład:	Jeśli akumulator zawiera 0C3h (11000011b), rejestr R0 zawiera 0AAh (10101010b), to instrukcja XRL A, R0 pozostawi w akumulatorze wartość 69h (01101001b). Instrukcję można wykorzystywać do negowania wybranych bitów wybranego bajtu wewnętrznej pamięci RAM lub rejestru – jeśli miejscem przeznaczenia wyniku jest bajt adresowany bezpośrednio. Maską określającą, które bity będą negowane jest wówczas stałą



Jeśli instrukcja jest wykorzystywana do zmiany stanu linii portu, to wartością użytą jako pierwotny stan portu będzie wartość odczytana z rejestru wyjściowego portu, a nie rzeczywisty stan wyprowadzeń portu.

wyszczególnioną w instrukcji lub zawartością akumulatora (zwykle uzyskaną z poprzedzających instrukcję obliczeń). Instrukcja

XRL P1, #00110001B

spowoduje zanegowanie bitów 5, 4 i 0 portu P1.

XRL A, Rn

Operacja: XRL

$A := A \text{ xor } R_n$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

0	1	1	0	1	n	n	n
---	---	---	---	---	---	---	---

XRL A, direct

Operacja: XRL

$A := A \text{ xor } (\text{direct})$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	1	0	0	1	0	1
direct							

XRL A, @Ri

Operacja: XRL

$A := A \text{ xor } [R_i]$

Liczba cykli: 1

Liczba bajtów: 1

Kod instrukcji:

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

XRL A, #data

Operacja: XRL

$A := A \text{ xor } \text{data}$

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:

0	1	1	0	0	1	0	0
data							

XRL direct, A

Operacja: XRL
(direct) := A xor (direct)

Liczba cykli: 1

Liczba bajtów: 2

Kod instrukcji:	0	1	1	0	0	0	1	0
direct								

XRL direct, #data

Operacja: XRL
(direct) := (direct) xor data

Liczba cykli: 2

Liczba bajtów: 3

Kod instrukcji:	0	1	1	0	0	0	1	1
direct								
data								

7.3. Skrócona lista instrukcji mikrokontrolerów rodziny '51

Tabela zawiera listę instrukcji mikrokontrolerów rodziny '51 przedstawioną w skróconej formie.

Mnemonik	Argument(y)	Operacja	Liczba bajtów	Liczba cykli maszynowych
Operacje arytmetyczne				
ADD	A, Rn	Suma zawartości rejestru i akumulatora	1	1
ADD	A, direct	Suma zmiennej bezpośredniej i akumulatora	2	1
ADD	A, @Ri	Suma zmiennej pośredniej i akumulatora	1	1
ADD	A, #data	Suma stałej i akumulatora	2	1
ADDC	A, Rn	Suma zawartości rejestru, akumulatora i bitu CY	1	1
ADDC	A, direct	Suma zmiennej bezpośredniej, akumulatora i bitu CY	2	1
ADDC	A, @Ri	Suma zmiennej pośredniej, akumulatora i bitu CY	1	1
ADDC	A, #data	Suma stałej, akumulatora i bitu CY	2	1
SUBB	A, Rn	Różnica zawartości akumulatora, rejestru i bitu CY	1	1
SUBB	A, direct	Różnica zawartości akumulatora, zmiennej bezpośredniej i bitu CY	2	1
SUBB	A, @Ri	Różnica zawartości akumulatora, zmiennej pośredniej i bitu CY	1	1

Mnemonik	Argument(y)	Operacja	Liczba bajtów	Liczba cykli maszynowych
SUBB	A, #data	Różnica zawartości akumulatora, stałej i bitu CY	2	1
INC	A	Inkrementacja akumulatora	1	1
INC	Rn	Inkrementacja zawartości rejestru	1	1
INC	direct	Inkrementacja zmiennej bezpośredniej	2	1
INC	@Ri	Inkrementacja zmiennej pośredniej	1	1
DEC	A	Dekrementacja akumulatora	1	1
DEC	Rn	Dekrementacja zawartości rejestru	1	1
DEC	direct	Dekrementacja zmiennej bezpośredniej	2	1
DEC	@Ri	Dekrementacja zmiennej pośredniej	1	1
INC	DPTR	Inkrementacja zawartości wskaźnika danych	1	2
MUL	AB	Iloczyn A i B	1	4
DIV	AB	Dzielenie całkowite A przez B	1	4
DA	A	Dziesiętna poprawka akumulatora	1	1
Operacje logiczne				
ANL	A, Rn	Funkcja AND zawartości akumulatora i rejestru	1	1
ANL	A, direct	Funkcja AND akumulatora i zmiennej bezpośredniej	2	1
ANL	A, @Ri	Funkcja AND akumulatora i zmiennej pośredniej	1	1
ANL	A, #data	Funkcja AND akumulatora i stałej	2	1
ANL	direct, A	Funkcja AND zmiennej bezpośredniej i akumulatora	2	1
ANL	direct, #data	Funkcja AND zmiennej bezpośredniej i stałej	3	2
ORL	A, Rn	Funkcja OR zawartości akumulatora i rejestru	1	1
ORL	A, direct	Funkcja OR akumulatora i zmiennej bezpośredniej	2	1
ORL	A, @Ri	Funkcja OR akumulatora i zmiennej pośredniej	1	1
ORL	A, #data	Funkcja OR akumulatora i stałej	2	1
ORL	direct, A	Funkcja OR zmiennej bezpośredniej i akumulatora	2	1
ORL	direct, #data	Funkcja OR zmiennej bezpośredniej i stałej	3	2
XRL	A, Rn	Funkcja EXOR zawartości akumulatora i rejestru	1	1
XRL	A, direct	Funkcja EXOR akumulatora i zmiennej bezpośredniej	2	1
XRL	A, @Ri	Funkcja EXOR akumulatora i zmiennej pośredniej	1	1
XRL	A, #data	Funkcja EXOR akumulatora i stałej	2	1

Mnemonic	Argument(y)	Operacja	Liczba bajtów	Liczba cykli maszynowych
XRL	direct, A	Funkcja EXOR zmiennej bezpośredniej i akumulatora	2	1
XRL	direct, #data	Funkcja EXOR zmiennej bezpośredniej i stałej	3	2
CLR	A	Zerowanie akumulatora	1	1
CPL	A	Negacja zawartości akumulatora	1	1
RL	A	Cykliczne przesunięcie zawartości akumulatora w lewo	1	1
RLC	A	Cykliczne przesunięcie w lewo przez bit CY	1	1
RR	A	Cykliczne przesunięcie zawartości akumulatora w prawo	1	1
RRC	A	Cykliczne przesunięcie w prawo przez bit CY	1	1
SWAP	A	Zamiana miejscami 4-bitowych połówek akumulatora	1	1
Operacje przesyłania danych				
MOV	A, Rn	Przesłanie zawartości rejestru do akumulatora	1	1
MOV	A, direct	Przesłanie zmiennej bezpośredniej do akumulatora	2	1
MOV	A, @Ri	Przesłanie zmiennej pośredniej do akumulatora	1	1
MOV	A, #data	Przesłanie stałej do akumulatora	2	1
MOV	Rn, A	Przesłanie zawartości akumulatora do rejestru	1	1
MOV	Rn, direct	Przesłanie zmiennej bezpośredniej do rejestru	2	2
MOV	Rn, #data	Przesłanie stałej do rejestru	2	1
MOV	direct, A	Przesłanie zawartości akumulatora do zmiennej bezpośredniej	2	1
MOV	direct, Rn	Przesłanie zawartości rejestru do zmiennej bezpośredniej	2	2
MOV	direct, direct	Przesłanie zmiennej bezpośredniej do zmiennej bezpośredniej	3	2
MOV	direct, @Ri	Przesłanie zmiennej pośredniej do zmiennej bezpośredniej	2	2
MOV	direct, #data	Przesłanie stałej do zmiennej bezpośredniej	3	2
MOV	@Ri, A	Przesłanie zawartości akumulatora do zmiennej pośredniej	1	1
MOV	@Ri, direct	Przesłanie zmiennej bezpośredniej do zmiennej pośredniej	2	2
MOV	@Ri, #data	Przesłanie stałej do zmiennej pośredniej	2	1
MOV	DPTR, #data16	Ładowanie wskaźnika danych stałą 16-bitową	3	2

Mnemonik	Argument(y)	Operacja	Liczba bajtów	Liczba cykli maszynowych
MOVC	A, @A+DPTR	Przesłanie do akumulatora bajtu kodu adresowanego indeksowo względem wskaźnika danych	1	2
MOVC	A, @A+PC	Przesłanie do akumulatora bajtu kodu adresowanego indeksowo względem licznika rozkazów	1	2
MOVX	A, @Ri	Przesłanie do akumulatora bajtu z zewnętrznej pamięci RAM (adresowanie 8-bitowe)	1	2
MOVX	A, @DPTR	Przesłanie do akumulatora bajtu z zewnętrznej pamięci RAM (adresowanie 16-bitowe)	1	2
MOVX	@Ri, A	Przesłanie zawartości akumulatora do zewnętrznej pamięci RAM (adresowanie 8-bitowe)	1	2
MOVX	@DPTR, A	Przesłanie zawartości akumulatora do zewnętrznej pamięci RAM (adresowanie 16-bitowe)	1	2
PUSH	direct	Przesłanie zmiennej bezpośredniej na stos	2	2
POP	direct	Pobranie zmiennej bezpośredniej ze stosu	2	2
XCH	A, Rn	Zamiana zawartości rejestru i akumulatora	1	1
XCH	A, direct	Zamiana zawartości akumulatora i zmiennej bezpośredniej	2	1
XCH	A, @Ri	Zamiana zawartości akumulatora i zmiennej pośredniej	1	1
XCHD	A, @Ri	Zamiana zawartości 4 młodszych bitów akumulatora i zmiennej pośredniej	1	1
Operacje na bitach				
CLR	C	Wyzerowanie bitu CY	1	1
CLR	bit	Wyzerowanie bitu adresowanego bezpośrednio	2	1
SETB	C	Ustawienie bitu CY	1	1
SETB	bit	Ustawienie bitu adresowanego bezpośrednio	2	1
CPL	C	Zanegowanie bitu CY	1	1
CPL	bit	Zanegowanie bitu adresowanego bezpośrednio	2	1
ANL	C, bit	Iloczyn logiczny bitu CY i bitu adresowanego bezpośrednio	2	2
ANL	C, /bit	Iloczyn logiczny bitu CY i negacji bitu adresowanego bezpośrednio	2	2
ORL	C, bit	Suma logiczna bitu CY i bitu adresowanego bezpośrednio	2	2
ORL	C, /bit	Suma logiczna bitu CY i negacji bitu adresowanego bezpośrednio	2	2
MOV	C, bit	Przesłanie bitu adresowanego bezpośrednio do CY	2	1

Mnemonik	Argument(y)	Operacja	Liczba bajtów	Liczba cykli maszynowych
Instrukcje sterujące				
ACALL	addr11	Bezwarunkowe wywołanie procedury	2	2
LCALL	addr16	Dalekie wywołanie procedury	3	2
RET		Powrót z procedury	1	2
RETI		Powrót z procedury obsługi przerwania	1	2
AJMP	addr11	Skok bezwzględny	2	2
LJMP	addr16	Skok długi	3	2
SJMP	rel	Krótki skok pod adres względny	2	2
JMP	@A + DPTR	Skok pośredni	1	2
JZ	rel	Skok, jeśli akumulator jest zerem	2	2
JNZ	rel	Skok, jeśli akumulator nie jest zerem	2	2
JC	rel	Skok, jeśli CY jest jedynką	2	2
JNC	rel	Skok, jeśli CY jest zerem	2	2
JB	bit, rel	Skok, jeśli bit jest jedynką	3	2
JNB	bit, rel	Skok, jeśli bit jest zerem	3	2
JBC	bit, rel	Skok, jeśli bit jest jedynką i wyzerowanie bitu	3	2
CJNE	A, direct, rel	Skok, jeśli zmienna bezpośrednia i akumulator są różne	3	2
CJNE	A, #data, rel	Skok, jeśli stała i zawartość akumulatora są różne	3	2
CJNE	Rn, #data, rel	Skok, jeśli stała i zawartość rejestru są różne	3	2
CJNE	@Ri, #data, rel	Skok, jeśli stała i zmienna pośrednia są różne	3	2
DJNZ	Rn, rel	Dekrementacja zawartości rejestru i skok, jeśli nie zero	2	2
DJNZ	direct, rel	Dekrementacja zmiennej bezpośredniej i skok, jeśli nie zero	3	2
NOP		Operacja pusta	1	1

Rejestr SFR



W mikrokontrolerach rodziny 8048 stanowiących pierwowzór dla rodziny '51 sterowanie wewnętrznymi układami mikrokontrolera (licznik, układ przerwań itp.) odbywało się za pomocą specjalnych instrukcji. W ramach listy instrukcji mikrokontrolerów rodziny 8048 występowały zatem oddzielne rozkazy powodujące uruchomienie i zatrzymanie pracy licznika, odczyt jego zawartości, zablokowanie, bądź odblokowanie poszczególnych przerwań itp. Takie rozwiązanie sposobu sterowania wewnętrznymi układami mikrokontrolera było jednak nie do przyjęcia w przypadku mikrokontrolerów o większych zasobach i możliwościach wewnętrznych układów peryferyjnych. Z tego powodu sterowanie wewnętrznymi układami mikrokontrolerów rodziny '51 realizowane jest nie za pomocą specjalizowanych instrukcji, lecz za pomocą zwykłych instrukcji odwołujących się do wyspecjalizowanych rejestrów. Stan tych rejestrów (określanych mianem rejestrów specjalnego przeznaczenia – w skrócie SFR) wpływa już bezpośrednio na pracę wewnętrznych układów mikrokontrolera. Podstawową funkcją rejestrów SFR występujących w mikrokontrolerach rodziny '51 jest zatem sterowanie pracą wewnętrznych układów peryferyjnych mikrokontrolera. Liczba rejestrów SFR i sposób ich oddziaływania na wewnętrzne układy peryferyjne zależą od typu mikrokontrolera.

Ponieważ już w mikrokontrolerze 8051 występowało ponad dwadzieścia rejestrów SFR, zostały one umieszczone w oddzielnym obszarze adresowym wewnętrznej pamięci RAM mikrokontrolera (patrz rozdz. 2.1). Ostatecznie, oprócz rejestrów pełniących rolę interfejsu między CPU a wewnętrznymi układami peryferyjnymi, w przestrzeni rejestrów SFR znalazły się także takie obiekty (wykorzystywane przede wszystkim przez CPU) jak akumulator ACC, wskaźnik stosu SP, wskaźnik danych DPTR, rejestr wskaźników stanu PSW itp.

8.1. Rozmieszczenie rejestrów SFR

Obszar adresowy rejestrów SFR obejmuje 128 bajtów wewnętrznej pamięci RAM (patrz rozdz. 2.1), ale w żadnym z dotychczas produkowanych mikrokontrolerów nie został on wykorzystany w całości. Niestety, pod adresami, które nie są zajęte przez rejestry SFR danego mikrokontrolera, nie są implementowane rejestry ogólnego zastosowania, czy inne zasoby pamięci. Tak więc wolne fragmenty obszaru adresowego rejestrów SFR nie mogą być używane do żadnych innych celów – próby zapisu lub odczytu informacji spod takich adresów będą dawały bliżej nieokreślone wyniki.

Na następnych stronach przedstawiono rozmieszczenie rejestrów SFR występujących w najczęściej stosowanych mikrokontrolerach rodziny '51. Warto zauważyć, że rejestry umieszczone w lewej skrajnej kolumnie kolejnych tabel mają adresy o wartościach podzielnych przez osiem, a w związku z tym są to rejestry dostępne dla operacji na pojedynczych bitach (patrz rozdz. 2.1.6). Pod nazwami poszczególnych rejestrów podane są (w zapisie binarnym) wartości przyjmowane przez te rejestry w wyniku operacji zerowania mikrokontrolera. Symbol X oznacza, że dany bit przyjmuje podczas zerowania mikrokontrolera wartość nieokreśloną. Nazwa rejestru ujęta w nawiasy oznacza, że dany rejestr nie występuje w mikrokontrolerach niektórych producentów.

Tab. 8.1. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach C51 (z wyjątkiem najnowszych wersji układów Philipsa – patrz rozmieszczenie rejestrów C58), LV51

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000								D7
C8									CF
C0									C7
B8	IP XXX00000								BF
B0	P3 11111111								B7
A8	IE 0XX00000								AF
A0	P2 11111111								A7
98	SCON 00000000	SBUF XXXXXXXX							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TLO 00000000	TL1 00000000	TH0 00000000	TH1 00000000	(AUXR) XXXXXX0		8F
80	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 0XX00000	87

Tab. 8.2. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach S51

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000								D7
C8									CF
C0									C7
B8	IP XX000000								BF
B0	P3 11111111								B7
A8	IE 0X000000								AF
A0	P2 11111111		AUXR1 XXXXXX0				WDRST XXXXXXX		A7
98	SCON 00000000	SBUF XXXXXXX							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XX00XX0		8F
80	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 00XX0000	87

Tab. 8.3. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach C52 (Intel), C54 (starsze wersje Philipsa)

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000								D7
C8	T2CON 00000000	(T2MOD) xxxxxx00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CF
C0									C7
B8	IP xx000000	(SADEN) 00000000							BF
B0	P3 11111111								B7
A8	IE 0x000000	(SADDR) 00000000							AF
A0	P2 11111111								A7
98	SCON 00000000	SBUF xxxxxxx0							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	(AUXR) xxxxxxx0		8F
80	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 00xx0000	87

Tab. 8.4. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach C52 (Atmel), LV52, C55

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000								D7
C8	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CF
C0									C7
B8	IP XX000000								BF
B0	P3 11111111								B7
A8	IE 0X000000								AF
A0	P2 11111111								A7
98	SCON 00000000	SBUF XXXXXXXX							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXXXXXXX0		8F
80	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 00XX0000	87

Uwaga! Rejestr AUXR w wielu mikrokontrolerach Atmela występuje jako niejawny, tzn. nie jest podawany w tabeli rejestrów SFR, lecz producent jednoznacznie przedstawia jego działanie w opisie sygnału ALE.

Tab. 8.5. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach S52

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000								D7
C8	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CF
C0									C7
B8	IP XX000000								BF
B0	P3 11111111								B7
A8	IE 0X000000								AF
A0	P2 11111111		AUXR1 XXXXXXX0				WDTRST XXXXXXX0		A7
98	SCON 00000000	SBUF XXXXXXXX							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TLO 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXX00XX0		8F
80	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 00XX0000	87

Tab. 8.6. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach S53, LS53

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000					SPCR 000001XX			D7
C8	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CF
C0									C7
B8	IP XX000000								BF
B0	P3 11111111								B7
A8	IE 0X000000		SPSR 00XXXXXX						AF
A0	P2 11111111								A7
98	SCON 00000000	SBUF XXXXXXXX							9F
90	P1 11111111						WCON 00000010		97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXXXXX0		8F
80	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000	SPDR XXXXXXXX	PCON 00XX0000	87

Uwaga! Rejestr AUXR w wielu mikrokontrolerach Atmela występuje jako niejawny, tzn. nie jest podawany w tabeli rejestrów SFR, lecz producent jednoznacznie przedstawia jego działanie w opisie sygnału ALE.

Tab. 8.7. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach C54 (Intel, nowsze wersje Philipsa), C58 i najnowszych wersjach C51 i C52 Philipsa

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000								D7
C8	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CF
C0									C7
B8	IP XX000000	SADEN 00000000							BF
B0	P3 11111111							IPH XX000000	B7
A8	IE 0X000000	SADDR 00000000							AF
A0	P2 11111111		(AUXR1) XX0000X0						A7
98	SCON 00000000	SBUF XXXXXXX							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXXXXXX0		8F
80	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 00X00000	87

Tab. 8.8. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach C1051

F8								FF
F0	B 00000000							F7
E8								EF
E0	ACC 00000000							E7
D8								DF
D0	PSW 00000000							D7
C8								CF
C0								C7
B8	IP XXXXX000							BF
B0	P3 11111111							B7
A8	IE 0XXXX000							AF
A0								A7
98								9F
90	P1 11111111							97
88	TCON 00000000	TMOD 00000000	TL0 00000000		TH0 00000000			8F
80		SP 00000111	DPL 00000000	DPH 00000000			PCON 0XXX0000	87

Tab. 8.9. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach C1051U, C2051, C4051

F8								FF
F0	B 00000000							F7
E8								EF
E0	ACC 00000000							E7
D8								DF
D0	PSW 00000000							D7
C8								CF
C0								C7
B8	IP xxx00000							BF
B0	P3 11111111							B7
A8	IE 0xx00000							AF
A0								A7
98	SCON 00000000	SBUF xxxxxxx						9F
90	P1 11111111							97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		8F
80		SP 00000111	DPL 00000000	DPH 00000000			PCON 0xxx0000	87

Tab. 8.10. Rozmieszczenie i stan początkowy (po wyzerowaniu mikrokontrolera) rejestrów SFR występujących w mikrokontrolerach S8252, LS8252

F8									FF
F0	B 00000000								F7
E8									EF
E0	ACC 00000000								E7
D8									DF
D0	PSW 00000000					SPCR 000001XX			D7
C8	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			C7
C0									C7
B8	IP XX000000								BF
B0	P3 11111111								B7
A8	IE 0X000000		SPSR 00000000						AF
A0	P2 11111111								A7
98	SCON 00000000	SBUF XXXXXXX							9F
90	P1 11111111						WMCON 00000010		97
88	TCON 00000000	TMOD 00000000	TLO 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXXXXXX0		8F
80	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000	SPDR XXXXXXX	PCON 0XXX0000	87

Uwaga! Rejestr AUXR w wielu mikrokontrolerach Atmela występuje jako niejawny, tzn. nie jest podawany w tabeli rejestrów SFR, lecz producent jednoznacznie przedstawia jego działanie w opisie sygnału ALE.

8.2. Opis rejestrów SFR

Specyficzna rola, jaką pełnią w mikrokontrolerach rodziny '51 rejestry SFR powoduje, że analiza zawartości i sposobu działania poszczególnych rejestrów może stanowić bogate źródło informacji o możliwościach funkcjonalnych i sposobie sterowania wewnętrznych układów peryferyjnych. Z tego właśnie względu znaczna część książki poświęcona została na dokładne omówienie rejestrów SFR występujących w najczęściej wykorzystywanych mikrokontrolerach rodziny '51.

W przedstawionym na następnych stronach opisie rejestrów SFR przyjęto następującą konwencję:

- a) rejestry są opisane w kolejności alfabetycznej z podaniem ich adresów i funkcji oraz oznaczeń mikrokontrolerów, w których rejestry te występują,
- b) rejestry o identycznych nazwach ale różnych adresach zostały opisane oddzielnie,
- c) zawartość rejestrów z podaniem nazw, znaczenia i sposobu funkcjonowania poszczególnych bitów jest przedstawiona w postaci tabel,
- d) z lewej strony tabel zawartości rejestrów są umieszczone bity najbardziej znaczące,
- e) linie przerywane w tabeli przedstawiającej rozmieszczenie bitów w rejestrze sygnalizują, że operacje na poszczególnych bitach mogą być realizowane wyłącznie przy użyciu instrukcji działających na pełnych bajtach, linie ciągle oznaczają, że rejestr zawiera bity indywidualnie adresowalne, a zatem do odczytu, bądź modyfikacji tych bitów mogą być wykorzystywane instrukcje bitowe,
- f) w przypadku występowania kilku wariantów zawartości rejestru o danej nazwie, opis rejestru obejmuje odpowiednią liczbę tabel (opatrzonych oznaczeniami mikrokontrolerów) przedstawiających rozmieszczenie bitów w poszczególnych wariantach rejestrów (opis wszystkich bitów ujęto w jednej tabeli),
- g) w przypadku modyfikacji nazw bitów (w zależności od typu mikrokontrolera) pełniących identyczne funkcje w danym rejestrze, bity te zostały opisane wspólnie, zmodyfikowane zaś nazwy bitów umieszczono w nawiasach okrągłych (bezpośrednio pod nazwą podstawową),
- h) stan początkowy (po wyzerowaniu mikrokontrolera) poszczególnych rejestrów zależy od typu mikrokontrolera i może być odczytany z tabel umieszczonych w rozdz. 8.1.

Wykaz rejestrów SFR mikrokontrolerów rodziny '51

ACC (0E0h).....	157
AUXR (8Eh).....	157
AUXR1 (0A2h)	157
B (0F0h)	159
DPH (83h)	159
DP0H (83h)	159
DP1H (85h).....	159
DPL (82h)	159
DP0L (82h)	159
DP1L (84h).....	159
IE (0A8h).....	160
IP (0B8h).....	160
IPH (0B7h)	161
P0 (80h)	161
P1 (90h)	161
P2 (0A0h)	162
P3 (0B0h)	162
PCON (87h)	162
PSW (0D0h)	163
RCAP2H (0CBh)	164
RCAP2L (0CAh).....	164
SADDR (0A9h).....	164
SADEN (0B9h).....	164
SBUF (99h).....	164
SCON (98h).....	165
SP (81h).....	166
SPCR (0D5h)	166
SPDR (086h)	167
SPSR (0AAh)	167
TCON (88h)	168
T2CON (0C8h)	169
TH0 (8Ch)	170
TH1 (8Dh).....	170
TH2 (0CDh)	170
TL0 (8Ah).....	170
TL1 (8Bh)	170
TL2 (0CCCh).....	170
TMOD (89h)	171
T2MOD (0C9h)	172
WCON (96h).....	172
WMCON (96h).....	173

ACC (0E0h) – akumulator
wszystkie mikrokontrolery rodziny '51

ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0
-------	-------	-------	-------	-------	-------	-------	-------

AUXR (8Eh) – rejestr dodatkowych bitów sterujących
C51 (Atmel, nowe wersje Philipsa), C52 (Atmel, nowe wersje Philipsa), LV52, S53, LS53, C54 (Intel, nowe wersje Philipsa), C55, S8252, LS8252

-	-	-	-	-	-	-	A0
---	---	---	---	---	---	---	----

S51, S52

-	-	-	WDIDLE	DISRTO	-	-	DISALE
---	---	---	--------	--------	---	---	--------

Bit	Funkcja
WDIDLE	Bit zezwolenia na pracę licznika czuwającego w trybie uśpienia: 0 – licznik czuwający zlicza także po przejściu mikrokontrolera w tryb uśpienia, 1 – w trybie uśpienia licznik czuwający jest zablokowany (nie zlicza).
DISRTO	Bit trybu pracy wyprowadzenia RST: 0 – jako wejście i wyjście sygnału zerowania mikrokontrolera (przy przepelnieniu licznika czuwającego na wyprowadzeniu generowany jest impuls zerowania), 1 – wyłącznie jako wejście.
DISALE	Bit sterowania pracą sygnału ALE: 0 – generowanie sygnału ALE z częstotliwością $f_{OSC}/6$, 1 – sygnał ALE generowany tylko podczas odwołań do zewnętrznej pamięci programu lub danych.
A0	Bit blokowania sygnału ALE: 0 – standardowe generowanie sygnału ALE, 1 – sygnał ALE generowany tylko podczas odwołań do zewnętrznej pamięci programu lub danych.

AUXR1 (0A2h) – rejestr dodatkowych bitów sterujących
S51, S52, nowe wersje 80C31 Philipsa

-	-	-	-	-	-	-	DPS
---	---	---	---	---	---	---	-----

nowe wersje 80C51 Philipsa

-	-	-	-	WUPD	0	-	DPS
---	---	---	---	------	---	---	-----

nowe wersje 87C51 Philipsa

–	–	–	LPEP	WUPD	0	–	DPS
---	---	---	------	------	---	---	-----

nowe wersje 87C52, 87C54, 87C58 Philipsa

–	–	–	LPEP	GF2	0	–	DPS
---	---	---	------	-----	---	---	-----

pozostałe nowe wersje C51, C52, C54 i C58 Philipsa (w szczególności układy z pamięcią programu typu Flash)

–	–	–	–	GF2	0	–	DPS
---	---	---	---	-----	---	---	-----

Bit	Funkcja
DPS	Bit określający numer aktywnego wskaźnika danych: 0 – DPTR0 (wskaźnik podstawowy), 1 – DPTR1 (wskaźnik dodatkowy). Wszystkie instrukcje wykonywane z udziałem wskaźnika DPTR (MOVX; INC DPTR; MOV DPTR, #stała) odnoszą się do wskaźnika aktywnego. W przypadku mikrokontrolerów firmy Atmel w każdej chwili możliwy jest bezpośredni dostęp do obu wskaźników DPTR (za pomocą rejestrów DP0H, DP0L, DP1H, DP1L). W przypadku układów firmy Philips, dostęp do starszej i młodszej połówki DPTR obydwu wskaźników odbywa się za pomocą tych samych adresów w przestrzeni rejestrów SFR (brak dodatkowych rejestrów DP1H, DP1L), zatem w danej chwili możliwy jest dostęp wyłącznie do aktywnego wskaźnika danych. Jeśli rejestr AUXR1 zawiera więcej bitów sterujących niż DPS, to bit 2. rejestru jest zawsze zerem, co pozwala na przełączanie wskaźników DPTR instrukcją INC AUXR1, która zmienia wówczas wyłącznie stan bitu DPS.
WUPD	Bit zezwolenia na wychodzenie z trybu zamrożenia za pomocą przerwań zewnętrznych: 0 – wyjście z zamrożenia za pomocą przerwań zewnętrznych zablokowane, 1 – wyjście z zamrożenia za pomocą przerwań zewnętrznych możliwe.
LPEP	Bit LPEP występuje w mikrokontrolerach wyposażonych w pamięć programu typu EPROM, które mogą pracować z zasilaniem niskonapięciowym. Ponieważ pamięć EPROM zawiera pewne układy analogowe, które można wyłączyć jeśli napięcie zasilania jest niższe niż 4,0 V (muszą być zasilane jeśli napięcie zasilające ma wartość wyższą niż 4,0 V), bit umożliwia wyłączenie tych dodatkowych układów analogowych, zmniejszając tym samym moc pobieraną przez mikrokontroler: 0 – standardowa praca pamięci EPROM, 1 – wyłączona część układów sterujących pamięcią EPROM (praca ze zmniejszonym poborem mocy, dozwolona tylko dla $V_{CC} < 4,0\text{ V}$).
GF2	Bit ogólnego zastosowania.

B (0F0h) – akumulator pomocniczy
wszystkie mikrokontrolery rodziny '51

B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
-----	-----	-----	-----	-----	-----	-----	-----

DPH (83h) – bardziej znaczący bajt wskaźnika danych DPTR
wszystkie mikrokontrolery rodziny '51 oprócz S51, S52, S53, LS53, S8252, LS8252

DPH.7	DPH.6	DPH.5	DPH.4	DPH.3	DPH.2	DPH.1	DPH.0
-------	-------	-------	-------	-------	-------	-------	-------

DP0H (83h) – bardziej znaczący bajt wskaźnika danych DPTR
S51, S52, S53, LS53, S8252, LS8252

DP0H.7	DP0H.6	DP0H.5	DP0H.4	DP0H.3	DP0H.2	DP0H.1	DP0H.0
--------	--------	--------	--------	--------	--------	--------	--------

DP1H (85h) – bardziej znaczący bajt dodatkowego wskaźnika danych DPTR
S51, S52, S53, LS53, S8252, LS8252

DP1H.7	DP1H.6	DP1H.5	DP1H.4	DP1H.3	DP1H.2	DP1H.1	DP1H.0
--------	--------	--------	--------	--------	--------	--------	--------

DPL (82h) – mniej znaczący bajt wskaźnika danych DPTR
wszystkie mikrokontrolery rodziny '51 oprócz S51, S52, S53, LS53, S8252, LS8252

DPL.7	DPL.6	DPL.5	DPL.4	DPL.3	DPL.2	DPL.1	DPL.0
-------	-------	-------	-------	-------	-------	-------	-------

DP0L (82h) – mniej znaczący bajt wskaźnika danych DPTR
S51, S52, S53, LS53, S8252, LS8252

DP0L.7	DP0L.6	DP0L.5	DP0L.4	DP0L.3	DP0L.2	DP0L.1	DP0L.0
--------	--------	--------	--------	--------	--------	--------	--------

DP1L (84h) – mniej znaczący bajt dodatkowego wskaźnika danych DPTR
S51, S52, S53, LS53, S8252, LS8252

DP1L.7	DP1L.6	DP1L.5	DP1L.4	DP1L.3	DP1L.2	DP1L.1	DP1L.0
--------	--------	--------	--------	--------	--------	--------	--------

IE (0A8h) – rejestr zezwoleń na przerwania
51, C51, LV51, S51, C1051U, C2051, C4051

EA	–	–	ES	ET1	EX1	ET0	EX0
----	---	---	----	-----	-----	-----	-----

nowsze wersje C51 Philipsa, 52, C52, LV52, S52, S53, LS53, C54, C55, C58, S8252, LS8252

EA	–	ET2	ES	ET1	EX1	ET0	EX0
----	---	-----	----	-----	-----	-----	-----

C1051

EA	–	–	–	–	EX1	ET0	EX0
----	---	---	---	---	-----	-----	-----

Bit	Funkcja
EA	Bit globalnego zezwolenia na przerwania. Przy EA = 0 wszystkie przerwania są zablokowane, bez względu na stan indywidualnych bitów zezwolenia na przerwania. Przy EA = 1 przerwania są blokowane/odblokowywane za pomocą indywidualnych bitów zezwoleń na przerwania: 0 – przerwanie zablokowane, 1 – przerwanie odblokowane.
ET2	Bit zezwolenia na przerwanie od układu licznikowego T2.
ES	Bit zezwolenia na przerwanie od łącza szeregowego.
ET1	Bit zezwolenia na przerwanie od układu licznikowego T1.
EX1	Bit zezwolenia na przerwanie zewnętrzne $\overline{\text{INT1}}$.
ET0	Bit zezwolenia na przerwanie od układu licznikowego T0.
EX0	Bit zezwolenia na przerwanie zewnętrzne $\overline{\text{INT0}}$.

IP (0B8h) – podstawowy rejestr priorytetów przerwań
51, C51, LV51, S51, C1051U, C2051, C4051

–	–	–	PS	PT1	PX1	PT0	PX0
---	---	---	----	-----	-----	-----	-----

nowsze wersje C51 Philipsa, 52, C52, LV52, S52, S53, LS53, C54, C55, C58, S8252, LS8252

–	–	PT2	PS	PT1	PX1	PT0	PX0
---	---	-----	----	-----	-----	-----	-----

C1051

–	–	–	–	–	PX1	PT0	PX0
---	---	---	---	---	-----	-----	-----

Bit	Funkcja
PT2	Bit priorytetu przerwania od układu licznikowego T2.
PS	Bit priorytetu przerwania od łącza szeregowego.
PT1	Bit priorytetu przerwania od układu licznikowego T1.
PX1	Bit priorytetu przerwania zewnętrznego $\overline{\text{INT1}}$.
PT0	Bit priorytetu przerwania od układu licznikowego T0.
PX0	Bit priorytetu przerwania zewnętrznego $\overline{\text{INT0}}$.

IPH (0B7h)

– rejestr bardziej znaczących bitów priorytetów przerwania

nowsze wersje układów C51, C52 Philipsa, C54 (Intel, nowsze wersje Philipsa), C58

–	–	PT2H	PSH	PT1H	PX1H	PT0H	PX0H
---	---	-------------	------------	-------------	-------------	-------------	-------------

Bit	Funkcja
PT2H	Bardziej znaczący bit priorytetu przerwania od układu licznikowego T2.
PSH	Bardziej znaczący bit priorytetu przerwania od łącza szeregowego.
PT1H	Bardziej znaczący bit priorytetu przerwania od układu licznikowego T1.
PX1H	Bardziej znaczący bit priorytetu przerwania zewnętrznego $\overline{\text{INT1}}$.
PT0H	Bardziej znaczący bit priorytetu przerwania od układu licznikowego T0.
PX0H	Bardziej znaczący bit priorytetu przerwania zewnętrznego $\overline{\text{INT0}}$.

P0 (80h)

– rejestr portu P0

wszystkie mikrokontrolery rodziny '51 w obudowach 40-końcówkowych lub większych (nie występuje zatem m.in. w C1051, C1051U, C2051, C4051)

P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
-------------	-------------	-------------	-------------	-------------	-------------	-------------	-------------

P1 (90h)

– rejestr portu P1

wszystkie mikrokontrolery rodziny '51


P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
-------------	-------------	-------------	-------------	-------------	-------------	-------------	-------------

P2 (0A0h) - rejestr portu P2
*wszystkie mikrokontrolery rodziny '51 w obudowach 40-końcówkowych lub większych
(nie występuje zatem m.in. w C1051, C1051U, C2051, C4051)*

P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
------	------	------	------	------	------	------	------

P3 (0B0h) - rejestr portu P3
wszystkie mikrokontrolery rodziny '51

P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
------	------	------	------	------	------	------	------

**UWAGA**

W mikrokontrolerach C1051 i C2051 nie występuje wyprowadzenie P3.6, zaś bit P3.6 rejestru P3 odzwierciedla stan wyjścia analogowego komparatora, którego wejścia są podłączone do wyprowadzeń P1.0 (+) i P1.1 (-) mikrokontrolera.

PCON (87h) - rejestr sterujący pracą mikrokontrolerów w trybach zmniejszonego poboru mocy
51, 52 (mikrokontrolery NMOS)

SMOD	-	-	-	-	-	-	-
------	---	---	---	---	---	---	---

C51, LV51, C52 (Atmel, Siemens itp.), LV52, C55, C1051U, C2051, C4051

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

nowsze wersje C51 i C52 Philipsa, C52(Intel), C54, C58

SMOD1	SMOD0	-	POF	GF1	GF0	PD	IDL
-------	-------	---	-----	-----	-----	----	-----

C1051

-	-	-	-	GF1	GF0	PD	IDL
---	---	---	---	-----	-----	----	-----

S51, S52, S53, LS53, S8252, LS8252

SMOD	-	-	POF	GF1	GF0	PD	IDL
------	---	---	-----	-----	-----	----	-----

Bit	Funkcja
OV	Wskaźnik przepełnienia. Stan wskaźnika zależy od wyniku ostatnio wykonanej instrukcji mnożenia, dzielenia, dodawania lub odejmowania.
F1	Bit ogólnego zastosowania.
P	Wskaźnik parzystości, ustawiany gdy liczba bitów akumulatora równych jeden jest parzysta, zerowany w przeciwnym przypadku.

RCAP2H (0CBh) – bardziej znaczący bajt rejestru przechwytyjącego (układu licznikowego T2)

52, C52, LV52, S52, S53, LS53, C54, C55, C58, S8252, LS8252 i nowsze wersje C51 Philipsa

RCAP2H.7	RCAP2H.6	RCAP2H.5	RCAP2H.4	RCAP2H.3	RCAP2H.2	RCAP2H.1	RCAP2H.0
----------	----------	----------	----------	----------	----------	----------	----------

RCAP2L (0CAh) – mniej znaczący bajt rejestru przechwytyjącego (układu licznikowego T2)

52, C52, LV52, S52, S53, LS53, C54, C55, C58, S8252, LS8252 i nowsze wersje C51 Philipsa

RCAP2L.7	RCAP2L.6	RCAP2L.5	RCAP2L.4	RCAP2L.3	RCAP2L.2	RCAP2L.1	RCAP2L.0
----------	----------	----------	----------	----------	----------	----------	----------

SADDR (0A9h) – właściwy rejestr maski adresowej rozszerzonego łącza szeregowego
nowsze wersje C51 Philipsa, C52 (Intel, nowsze wersje Philipsa), C54, C58

SADDR.7	SADDR.6	SADDR.5	SADDR.4	SADDR.3	SADDR.2	SADDR.1	SADDR.0
---------	---------	---------	---------	---------	---------	---------	---------

SADEN (0B9h) – pomocniczy rejestr maski adresowej rozszerzonego łącza szeregowego

nowsze wersje C51 Philipsa, C52 (Intel, nowsze wersje Philipsa), C54, C58

SADEN.7	SADEN.6	SADEN.5	SADEN.4	SADEN.3	SADEN.2	SADEN.1	SADEN.0
---------	---------	---------	---------	---------	---------	---------	---------

SBUF (99h) – rejestr danych łącza szeregowego

51, C51, LV51, S51, 52, C52, LV52, S52, S53, LS53, C54, C55, C58, C1051U, C2051, C4051, S8252, LS8252

SBUF.7	SBUF.6	SBUF.5	SBUF.4	SBUF.3	SBUF.2	SBUF.1	SBUF.0
--------	--------	--------	--------	--------	--------	--------	--------

SCON (98h) – rejestr sterujący łącza szeregowego.

51, C51, LV51, S51, 52, C52 (Siemens, Atmel itp.), LV52, S52, S53, LS53, C55, C1051U, C2051, C4051, S8252, LS8252

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

nowsze wersje C51 i C52 Philipsa, C52 (Intel), C54, C58

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

Bit	Funkcja
SM0/FE	Znaczenie bitu zależy od stanu bitu SMOD0 w rejestrze PCON. Jeśli SMOD0 = 0, to bit SM0/FE funkcjonuje jako bit SM0 opisany poniżej i służy do ustalania trybu pracy łącza szeregowego. Dla SMOD0 = 1 bit SM0/FE funkcjonuje jako bit FE, będący wskaźnikiem braku bitu stopu odebranych danych. Bit FE jest ustawiany sprzętowo w wyniku braku bitu stopu, zerowany jest wyłącznie programowo.
SM0	Bity określające tryb pracy łącza szeregowego: SM0 SM1 0 0 – tryb 0, transmisja synchroniczna, 0 1 – tryb 1, transmisja asynchroniczna z możliwością regulacji prędkości transmisji, 8 bitów danych, 1 0 – tryb 2, transmisja asynchroniczna bez możliwości regulacji prędkości transmisji, 9 bitów danych, 1 1 – tryb 3, transmisja asynchroniczna z możliwością regulacji prędkości transmisji, 9 bitów danych.
SM1	
SM2	Bit sterujący przejściem w tryb komunikacji wieloprocesorowej. Dla SM2 = 1 dane są odbierane, a bit RI ustawiany tylko wtedy, jeśli 9. bit odebranych danych (bit stopu w przypadku przesyłania 8 bitów) jest jedynką. Niektórzy producenci zalecają, by przy transmisji synchronicznej (tryb 0 pracy łącza) bit SM2 był w stanie niskim (dla większości stan bitu SM2 podczas transmisji synchronicznej jest bez znaczenia).
REN	Bit zezwolenia na odbiór danych przez łącze szeregowe, ustawiany i zerowany programowo: 0 – transmisja ignorowana, 1 – dane odbierane.
TB8	Dodatkowy 9. bit danych nadawany podczas pracy łącza w trybie 2 lub 3, ustawiany i zerowany programowo.
RB8	Dodatkowy 9. bit danych odebranych przez łącze pracujące w trybie 2 lub 3. W przypadku pracy łącza szeregowego w trybie 1 i SM2 = 0, w RB8 umieszczany jest odebrany bit stopu. Bit RB8 nie jest używany podczas pracy łącza w trybie 0.
TI	Wskaźnik przerwania od nadajnika łącza szeregowego, ustawiany przy końcu nadawania 8. bitu danych (dla transmisji w trybie 0) lub na początku nadawania bitu stopu (w pozostałych trybach pracy łącza). Bit ustawiany jest podczas każdej transmisji i musi być zerowany programowo.

Bit	Funkcja
RI	Wskaźnik przerwania od odbiornika łącza szeregowego, ustawiany przy końcu odbierania 8. bitu danych (dla transmisji w trybie 0) lub w trakcie odbierania bitu stopu (w pozostałych trybach pracy łącza). Bit ustawiany jest podczas każdej transmisji i musi być zerowany programowo.

SP (81h) – wskaźnik stosu
wszystkie mikrokontrolery rodziny '51

SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
-----	-----	-----	-----	-----	-----	-----	-----

SPCR (0D5h) – rejestr sterujący interfejsu SPI
S53, LS53, S8252, LS8252

SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
------	-----	------	------	------	------	------	------

Bit	Funkcja
SPIE	Bit zezwolenia na przerwania od interfejsu SPI: 0 – przerwania od interfejsu SPI zablokowane, 1 – przerwania od interfejsu SPI odblokowane (tylko wtedy, gdy ustawiony jest bit ES w rejestrze EI).
SPE	Bit zezwolenia na pracę interfejsu SPI. Ustawienie bitu SPE aktywizuje interfejs SPI i powoduje, że linie P1.4...P1.7 mikrokontrolera zaczynają funkcjonować jako linie \overline{SS} , MOSI, MISO i SCK. Wyzerowanie bitu SPE blokuje pracę interfejsu.
DORD	Bit określający kolejność bitów w przesyłanych interfejsem SPI bajtach danych: 0 – dane wysyłane są począwszy od najbardziej znaczącego bitu, 1 – dane wysyłane są począwszy od najmniej znaczącego bitu.
MSTR	Bit wyboru trybu pracy interfejsu SPI: 0 – praca w trybie urządzenia podrzędnego, 1 – praca w trybie urządzenia nadrzędnego.
CPOL	Bit określający polaryzację sygnału taktującego interfejsu SPI: 0 – stanem nieaktywnym sygnału taktującego jest stan niski, 1 – stanem nieaktywnym sygnału taktującego jest stan wysoki.
CPHA	Bit określający przesunięcie fazy sygnału taktującego interfejsu SPI w stosunku do przesyłanych bitów danych (patrz też rys. 3.13 w rozdz. 3): 0 – próbkowanie stanu bitów odbieranych odbywa się przy przechodzeniu sygnału taktującego w stan aktywny, 1 – próbkowanie stanu bitów odbieranych odbywa się przy powrocie sygnału taktującego do stanu nieaktywnego.

Bit	Funkcja
SPR1	<p>Bity określające prędkość transmisji danych po interfejsie SPI.</p> <p>SPR1 SPR0</p> <p>0 0 - 4 cykle zegarowe mikrokontrolera na bit danych,</p> <p>0 1 - 16 cykli zegarowych mikrokontrolera na bit danych,</p>
SPR0	<p>1 0 - 64 cykle zegarowe mikrokontrolera na bit danych,</p> <p>1 1 - 128 cykli zegarowych mikrokontrolera na bit danych.</p> <p>Stan bitów jest bez znaczenia, jeśli interfejs SPI danego mikrokontrolera pracuje w trybie urządzenia podrzędnego.</p>

SPDR (086h) - rejestr danych interfejsu SPI występującego w mikrokontrolerach S53, LS53, S8252, LS8252

SPDR.7	SPDR.6	SPDR.5	SPDR.4	SPDR.3	SPDR.2	SPDR.1	SPDR.0
--------	--------	--------	--------	--------	--------	--------	--------

SPSR (0AAh) - rejestr stanu interfejsu SPI
S53, LS53, S8252, LS8252

SPIF	WCOL	-	-	-	-	-	-
------	------	---	---	---	---	---	---

Bit	Funkcja
SPIF	Wskaźnik przerwania od interfejsu SPI, ustawiany po zakończeniu każdej transmisji danych. Przerwanie jest zgłaszane, jeśli ustawione są bity ES (w rejestrze EI) i SPIE (w rejestrze SPCR). Bit SPIF jest zerowany sprzętowo w wyniku wykonania instrukcji zapisu lub odczytu rejestru SPDR poprzedzonej odczytem rejestru SPSR.
WCOL	Bit sygnalizujący błąd kolizji danych przeznaczonych do wysłania, ustawiany w wyniku próby zapisu nowych danych do rejestru SPDR podczas trwania (przed zakończeniem) transmisji po interfejsie SPI. Zapis danych do rejestru SPDR powodujący ustawienie bitu WCOL jest ignorowany (nie odnosi żadnego skutku). Bit WCOL jest zerowany sprzętowo w wyniku wykonania instrukcji zapisu lub odczytu rejestru SPDR poprzedzonej odczytem rejestru SPSR.

TCON (88h)

– rejestr sterujący układów licznikowych T0 i T1

C1051

–	–	TF	TR	IE1	IT1	IE0	IT0
---	---	----	----	-----	-----	-----	-----

pozostałe omawiane w książce mikrokontrolery rodziny '51

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

Bit	Funkcja
TF	Wskaźnik przerwania od licznika T0. Bit ustawiany sprzętowo wskutek przepełnienia starszego bajtu (TH) licznika T0, zerowany sprzętowo w chwili rozpoczęcia wykonywania procedury obsługi przerwania.
TR	Bit włączający (1) i wyłączający (0) układ licznikowy T0. Zerowany i ustawiany programowo.
TF1	Wskaźnik przerwania od licznika T1. Bit ustawiany sprzętowo wskutek przepełnienia licznika T1, zerowany sprzętowo w chwili rozpoczęcia wykonywania procedury obsługi przerwania.
TR1	Bit włączający (1) i wyłączający (0) układ licznikowy T1. Zerowany i ustawiany programowo.
TF0	Wskaźnik przerwania od licznika T0. Bit ustawiany sprzętowo wskutek przepełnienia licznika T0, zerowany sprzętowo w chwili rozpoczęcia wykonywania procedury obsługi przerwania.
TR0	Bit włączający (1) i wyłączający (0) układ licznikowy T0. Zerowany i ustawiany programowo.
IE1	Wskaźnik przerwania zewnętrznego $\overline{\text{INT1}}$. Ustawiany sprzętowo w wyniku wykrycia opadającego zbocza/niskiego poziomu na wyprowadzeniu $\overline{\text{INT1}}$ mikrokontrolera. Zerowany sprzętowo (tylko jeśli przerwanie jest aktywizowane zboczem sygnału) w wyniku rozpoczęcia wykonywania procedury obsługi przerwania. Jeśli przerwanie jest aktywizowane poziomem, stan bitu odzwierciedla stan wyprowadzenia mikrokontrolera.
IT1	Bit określający rodzaj przerwania zgłaszanego linią $\overline{\text{INT1}}$: 0 – przerwanie od niskiego poziomu na linii $\overline{\text{INT1}}$, 1 – przerwanie od opadającego zbocza na linii $\overline{\text{INT1}}$.
IE0	Wskaźnik przerwania zewnętrznego $\overline{\text{INT0}}$. Ustawiany sprzętowo w wyniku wykrycia opadającego zbocza/niskiego poziomu na wyprowadzeniu $\overline{\text{INT0}}$ mikrokontrolera. Zerowany sprzętowo (tylko jeśli przerwanie jest aktywizowane zboczem sygnału) w wyniku rozpoczęcia wykonywania procedury obsługi przerwania. Jeśli przerwanie jest aktywizowane poziomem, stan bitu odzwierciedla stan wyprowadzenia mikrokontrolera.
IT0	Bit określający rodzaj przerwania zgłaszanego linią $\overline{\text{INT0}}$: 0 – przerwanie od niskiego poziomu na linii $\overline{\text{INT0}}$, 1 – przerwanie od opadającego zbocza na linii $\overline{\text{INT0}}$.

TH0 (8Ch) – bardziej znaczący bajt układu licznikowego T0
wszystkie mikrokontrolery rodziny '51

TH0.7	TH0.6	TH0.5	TH0.4	TH0.3	TH0.2	TH0.1	TH0.0
-------	-------	-------	-------	-------	-------	-------	-------

TH1 (8Dh) – bardziej znaczący bajt układu licznikowego T1
wszystkie mikrokontrolery rodziny '51 z wyjątkiem C1051

TH1.7	TH1.6	TH1.5	TH1.4	TH1.3	TH1.2	TH1.1	TH1.0
-------	-------	-------	-------	-------	-------	-------	-------

TH2 (0CDh) – bardziej znaczący bajt układu licznikowego T2
52, C52, LV52, S52, S53, LS53, C54, C55, C58, S8252, LS8252 i nowsze wersje C51 Philipsa

TH2.7	TH2.6	TH2.5	TH2.4	TH2.3	TH2.2	TH2.1	TH2.0
-------	-------	-------	-------	-------	-------	-------	-------

TL0 (8Ah) – mniej znaczący bajt układu licznikowego T0
wszystkie mikrokontrolery rodziny '51

TL0.7	TL0.6	TL0.5	TL0.4	TL0.3	TL0.2	TL0.1	TL0.0
-------	-------	-------	-------	-------	-------	-------	-------

TL1 (8Bh) – mniej znaczący bajt układu licznikowego T1
wszystkie mikrokontrolery rodziny '51 z wyjątkiem C1051

TL1.7	TL1.6	TL1.5	TL1.4	TL1.3	TL1.2	TL1.1	TL1.0
-------	-------	-------	-------	-------	-------	-------	-------

TL2 (0CCh) – mniej znaczący bajt układu licznikowego T2
52, C52, LV52, S52, S53, LS53, C54, C55, C58, S8252, LS8252 i nowsze wersje C51 Philipsa

TL2.7	TL2.6	TL2.5	TL2.4	TL2.3	TL2.2	TL2.1	TL2.0
-------	-------	-------	-------	-------	-------	-------	-------

TMOD (89h) – rejestr sterujący trybem pracy układów licznikowych T0 i T1
C105I

-	-	-	-	GATE	C/ \bar{T}	M1	M0
---	---	---	---	------	--------------	----	----

pozostałe omawiane w książce mikrokontrolery rodziny '51

GATE	C/ \bar{T}	M1	M0	GATE	C/ \bar{T}	M1	M0
------	--------------	----	----	------	--------------	----	----

Bit	Funkcja
TMOD.7(GATE)	Bit sterujący bramkowaniem licznika T1. Przy ustawionym bicie GATE zliczanie następuje tylko wtedy, gdy zarówno linia $\overline{INT1}$, jak i bit TR1 (rejestr TCON) są w stanie wysokim. Jeśli bit GATE jest zerem, zliczanie jest sterowane wyłącznie stanem bitu TR1.
TMOD.6(C/\bar{T})	Bit określający źródło impulsów zliczanych przez układ licznikowy T1: 0 – zegar systemowy (zliczanie cykli maszynowych), 1 – wyprowadzenie T1 mikrokontrolera (impulsy zewnętrzne).
TMOD.5(M1)	Bity wyboru trybu pracy układu licznikowego T1: M1 M0 0 0 – tryb 0 (licznik 8-bitowy z 5-bitowym preskalerem), 0 1 – tryb 1 (licznik 16-bitowy), 1 0 – tryb 2 (licznik 8-bitowy z automatycznym przeładowywaniem w wyniku przepełnienia), 1 1 – tryb 3 (zatrzymanie zliczania).
TMOD.4(M0)	
TMOD.3(GATE)	
TMOD.2(C/\bar{T})	
TMOD.1(M1)	Bity wyboru trybu pracy układu licznikowego T0: M1 M0 0 0 – tryb 0 (licznik 8-bitowy z 5-bitowym preskalerem), 0 1 – tryb 1 (licznik 16-bitowy), 1 0 – tryb 2 (licznik 8-bitowy z automatycznym przeładowywaniem w wyniku przepełnienia), 1 1 – tryb 3 (dwa liczniki 8-bitowe).
TMOD.0(M0)	

T2MOD (0C9h) – rejestr sterujący trybem pracy układu licznikowego T2
C52, LV52, S52, S53, LS53, C54, C55, C58, S8252, LS8252 i nowsze wersje C51 Philipsa

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

C52 (Intel, Siemens itp.)

-	-	-	-	-	-	-	DCEN
---	---	---	---	---	---	---	------

Bit	Funkcja
T2OE	Bit zezwolenia na wykorzystanie wyprowadzenia T2 mikrokontrolera jako wyjścia o programowanej częstotliwości: 0 - wyprowadzenie T2 funkcjonuje jako wejście zliczanych impulsów lub jako zwykła linia wejść/wyjść cyfrowych, 1 - wyprowadzenie T2 jest wyjściem sygnału o programowanej częstotliwości.
DCEN	Bit określający kierunek zliczania licznika T2: 0 - zliczanie w górę, 1 - zliczanie w dół.

WCON (96h) – rejestr sterujący licznika czuwającego
S53, LS53

PS2	PS1	PS0	-	-	DPS	WDTRST	WDTEN
-----	-----	-----	---	---	-----	--------	-------

Bit	Funkcja
DPS	Bit wyboru aktywnego wskaźnika danych DPTR: 0 - aktywny jest wskaźnik podstawowy (DPTR0), 1 - aktywny jest wskaźnik dodatkowy (DPTR1).
WDTRST	Bit odświeżania licznika czuwającego. Skutkiem zapisu jedynki w miejscu bitu WDTRST jest odświeżenie licznika czuwającego, po którym bit WDTRST jest automatycznie zerowany. Bit WDTRST jest bitem przeznaczonym wyłącznie do zapisu.
WDTEN	Bit zezwolenia na pracę licznika czuwającego: 0 - licznik czuwający wyłączony, 1 - licznik czuwający aktywny. Licznik czuwający jest zatrzymywany na czas trybu zamrożenia. Bit WDTEN jest zerowany podczas operacji zerowania mikrokontrolera następującej po załączeniu napięcia zasilania.

Bit	Funkcja
PS2	Oznaczając 3-bitową liczbę binarną zakodowaną na bitach PS2...PS0 jako PS, czas T, po którym następuje przepełnienie licznika czuwającego można określić (z dokładnością uwzględniającą tolerancję częstotliwości generatora) z zależności: $T = 16 \cdot 2^{PS}$ Źródłem taktowania licznika czuwającego jest wewnętrzny niezależny generator, a tolerancja częstotliwości tego generatora przy napięciu zasilania 5,0 V wynosi $\pm 30\%$.
PS1	
PS0	

WMCON (96h) – rejestr sterujący licznika czuwającego i pamięci danych typu EEPROM.

S8252, LS8252 (zapis)

PS2	PS1	PS0	EEMWE	EEMEN	DPS	WDTRST	WDTEN
-----	-----	-----	-------	-------	-----	--------	-------

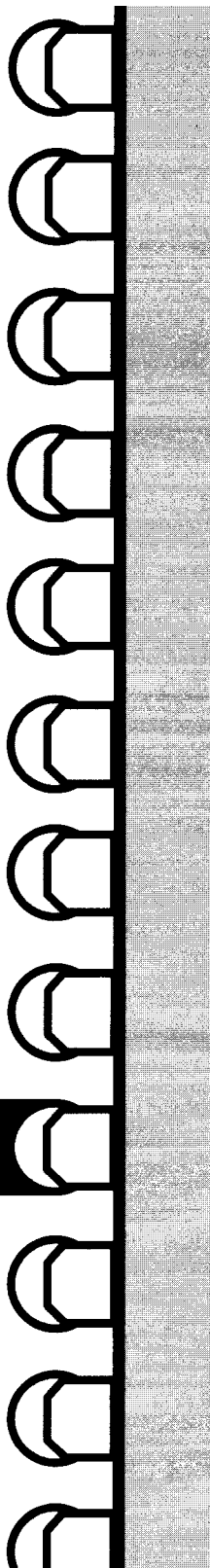
S8252, LS8252 (odczyt)

PS2	PS1	PS0	EEMWE	EEMEN	DPS	RDY/BSY	WDTEN
-----	-----	-----	-------	-------	-----	---------	-------

Bit	Funkcja
DPS	Bit wyboru aktywnego wskaźnika danych DPTR: 0 – aktywny jest wskaźnik podstawowy (DPTR0), 1 – aktywny jest wskaźnik dodatkowy (DPTR1).
WDTRST	Bit odświeżania licznika czuwającego. Skutkiem zapisu jedynki w miejscu bitu WDTRST jest odświeżenie licznika czuwającego, po którym bit WDTRST jest automatycznie zerowany.
RDY/BSY	Wskaźnik stanu pamięci EEPROM: 0 – pamięć jest w trakcie operacji programowania, 1 – możliwe jest rozpoczęcie nowej operacji programowania.
WDTEN	Bit zezwolenia na pracę licznika czuwającego: 0 – licznik czuwający wyłączony, 1 – licznik czuwający aktywny. Licznik czuwający jest zatrzymywany na czas trybu zamrożenia. Bit WDTEN jest zerowany podczas operacji zerowania mikrokontrolera następującej po załączeniu napięcia zasilania.
EEMWE	Bit zezwolenia na zapis do wewnętrznej pamięci EEPROM. Bit EEMWE musi być programowo ustawiony przed wykonaniem instrukcji typu MOVX powodującej zapis nowej wartości do pamięci EEPROM. Po zakończeniu operacji programowania (do testowania stanu pamięci EEPROM można wykorzystać bit RDY/BSY) bit EEMWE powinien być programowo wyzerowany.

Bit	Funkcja
EEMEN	Bit zezwolenia na dostęp do pamięci EEPROM: 0 - dostęp do pamięci EEPROM zablokowany, wszystkie operacje z użyciem wskaźnika DPTR odnoszą się do zewnętrznej pamięci danych, 1 - dostęp do pamięci EEPROM odblokowany, operacje z użyciem wskaźnika DPTR odnoszą się do wewnętrznej pamięci EEPROM.
PS2	Oznaczając 3-bitową liczbę binarną zakodowaną na bitach PS2...PS0 jako PS, czas T, po którym następuje przepełnienie licznika czuwającego można określić (z dokładnością uwzględniającą tolerancję częstotliwości generatora) z zależności: $T = 16 \cdot 2^{PS}$ Źródłem taktowania licznika czuwającego jest wewnętrzny niezależny generator, a tolerancja częstotliwości tego generatora przy napięciu zasilania 5,0 V wynosi $\pm 30\%$.
PS1	
PS0	

Wybrane narzędzia uruchomieniowe



Najprostszego nawet systemu mikroprocesorowego nie da się uruchomić bez użycia podstawowych narzędzi służących do tworzenia oprogramowania i uruchamiania systemów mikroprocesorowych. W przypadku urządzeń realizowanych z wykorzystaniem mikrokontrolerów jednoukładowych, narzędziami używanymi najczęściej (głównie z racji dostępności) są asemblery, kompilatory języków wysokiego poziomu, symulatory i debuggery wewnątrzukładowe. W przypadku najbardziej popularnych mikrokontrolerów, do jakich bez wątpienia należy zaliczyć układy rodziny '51, znaczna liczba narzędzi programowych oraz opisy niektórych narzędzi sprzętowych jest oferowana jako freeware lub shareware. Nie wszystkie one są wprawdzie tak dopracowane i tak funkcjonalne, jak większość produktów komercyjnych, jednak, zwłaszcza w przypadku amatorów, kwota kilku tysięcy złotych, jaką trzeba byłoby wydać na oprogramowanie komercyjne odstrasza z reguły na tyle skutecznie, że chętnie godzą się oni na mniej funkcjonalne, ale za to zdecydowanie tańsze opracowania ściągane z Internetu. Warto przy tym zauważyć, że w wersji freeware'owej są dostępne nie tylko domowe samoróbki, ale także niektóre pakiety komercyjne (tyle że w wersjach ograniczonych funkcjonalnie – np. z ograniczeniem maksymalnego rozmiaru kodu tworzonego w wyniku kompilacji).

9.1. Asemblery

W przypadku mikrokontrolerów podstawowym i najczęściej stosowanym narzędziem służącym do tworzenia oprogramowania (firmware'u) są asemblery. Ich główną zaletą jest możliwość pełnej optymalizacji wykorzystania wszelkich zasobów mikrokontrolera (przede wszystkim w sensie zajętości pamięci i szybkości wykonywania wygenerowanego kodu), a podstawową wadą – małą przejrzystość kodu. Z tego względu programowanie w assemblerze stosuje się zwykle tam, gdzie istotne jest uzyskanie maksymalnej efektywności kodu. W pozostałych przypadkach zdecydowanie chętniej jest wykorzystywane programowanie w językach wysokiego poziomu, które pozwala na znacznie szybsze tworzenie oprogramowania m.in. z racji wyposażenia w biblioteki standardowych procedur (np. arytmetycznych), większej przejrzystości kodu źródłowego i skuteczniejszego wspomagania w uruchamianiu oprogramowania (choćabyż ze względu na wychwytywanie znacznej liczby błędów już na etapie kompilacji).



Bezpłatne asemblery dla mikrokontrolerów '51 są dostępne w Internecie m.in. pod adresami:

- <http://plit.de/asem-51/download.htm> – ASEM51 w wersjach dla Windows, Linuxa i DOS,
- ftp://212.234.185.250/demo/51/kit51_728_exe – kompletne, bezpłatne środowisko projektowe dla assemblera '51 oraz C (z ograniczeniem objętości kodu do 4KB) firmy Rasionance,
- <http://www.xs4all.nl/~sbp/sbasmb/sba20825.zip> – SB-Assembler, jeden z lepszych bezpłatnych assemblerów z doskonałą dokumentacją,
- http://www.hw-server.com/x51_assemblers.html – zbiór linków do bezpłatnych assemblerów.

Liczba dostępnych na rynku assemblerów (w sensie oprogramowania tłumaczącego kod źródłowy napisany w assemblerze na kod maszynowy) dla mikrokontrolerów rodziny '51 jest liczona co najmniej w dziesiątkach i nie ma większego sensu dokonywanie w tym miejscu szczegółowego przeglądu tych programów, zwłaszcza że większość z nich działa co najmniej przyzwoicie. Warto natomiast zwrócić uwagę na kilka cech, jakimi dobrze jest się kierować przy wyborze konkretnego assemblera, ponieważ będą one miały wpływ na późniejszą funkcjonalność tego narzędzia. Zwłaszcza w przypadku tworzenia większych programów wygodnie jest, jeśli wykorzystywany assembler:

- a) umożliwia tworzenie oprogramowania z podziałem na moduły (zamiast pojedynczego pliku źródłowego z całym programem),
- b) ma wbudowany preprocesor obsługujący makrooperacje, translację warunkową itp.,
- c) stosuje powszechnie przyjęte (lub co najmniej często stosowane) notacje i konwencje składni (można np. znaleźć assembly, w których znakiem sygnalizującym początek komentarza jest wykrzyknik, a nie średnik).

9.2. Kompilatory języków wysokiego poziomu

Efektywność tworzenia kodu w assemblerze jest wielokrotnie niższa, niż w przypadku języków wysokiego poziomu, stąd też znaczna część oprogramowania systemów mikroprocesorowych tworzona jest w językach wysokiego poziomu. Nie inaczej jest też w przypadku układów rodziny '51.

Na rynku dostępnych jest wiele kompilatorów języków wysokiego poziomu na kod maszynowy mikrokontrolerów rodziny '51. Najbardziej popularne są kompilatory języka C, można jednak znaleźć także kompilatory Basica, Pascala, Moduli, czy nawet Fortha. Spośród opracowań komercyjnych przeznaczonych zwłaszcza do zastosowań profesjonalnych najczęściej wymienia się kompilatory języka C firm Keil, Tasking i Raisonance.

Obecnie zdecydowana większość komercyjnych pakietów kompilatorów oferuje standardowo:

- a) środowisko do zarządzania projektem, umożliwiające m.in. definiowanie i wielo-okienkową edycję zarówno plików z kodami źródłowymi, jak i plików dodatkowych (np. plików dokumentujących projekt),

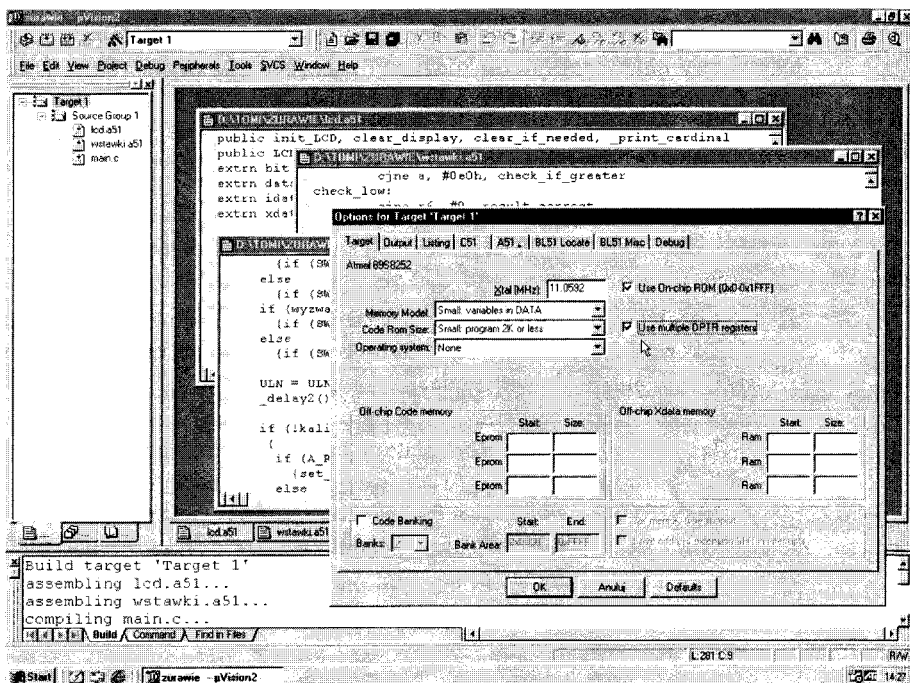


Bezpłatne kompilatory języków wysokiego poziomu dla mikrokontrolerów '51 są dostępne w Internecie m.in. pod adresami:

- http://www.hw-server.com/x51_ccompilers.html - zbiór linków do bezpłatnych kompilatorów języka C dla mikrokontrolerów '51,
- <http://www.ustr.net/files/files/forth51.zip> - kompilator języka Forth,
- http://www.hw-server.com/x51_pascals.html - zbiór linków do bezpłatnych kompilatorów języka Pascal dla mikrokontrolerów '51.

- b) predefiniowane nazwy rejestrów SFR dla większości dostępnych na rynku mikrokontrolerów,
- c) bogate biblioteki procedur standardowych obejmujących nie tylko najbardziej typowe procedury (np. zdefiniowane w standardzie ANSI C), lecz często także specjalizowane procedury obsługi wewnętrznych i zewnętrznych układów peryferyjnych (np. szyny I²C, przetworników A/C),
- d) dodatkowe opcje i funkcje pozwalające wykorzystać specyficzne cechy danego typu mikrokontrolera (np. specjalizowane pod kątem konkretnych mikrokontrolerów procedury biblioteczne wykorzystujące wewnętrzne sprzętowe układy arytmetyki wielokrotnej precyzji, optymalizacja kompilacji uwzględniająca możliwość korzystania z dodatkowych wskaźników DPTR – patrz np. rys. 9.1),
- e) możliwość programowania hybrydowego w języku wysokiego poziomu i w assemblerze,
- f) wbudowany symulator mikrokontrolera (zazwyczaj bardzo rozbudowany, oferujący bardzo silne wsparcie w uruchamianiu oprogramowania) oraz interfejsy programowe do innych narzędzi (emulatorów, debuggerów wewnątrzukładowych, programatorów),
- g) jądro wielozadaniowego systemu czasu rzeczywistego (nierzadko w kilku wersjach) umożliwiające stosunkowo łatwą implementację na mikrokontrolerze obsługi procesów niezależnych.

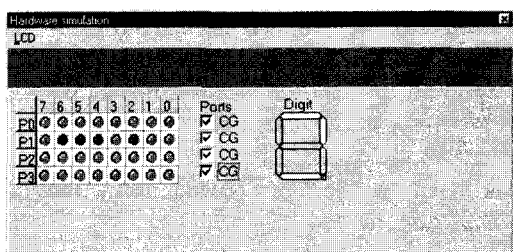
Spośród komercyjnych pakietów kompilatorów dla mikrokontrolerów rodziny '51 na szczególną uwagę zasługuje Bascom-8051 firmy MCS Electronics, będący



Rys. 9.1. Okno konfiguracji opcji pakietu Keil μ Vision



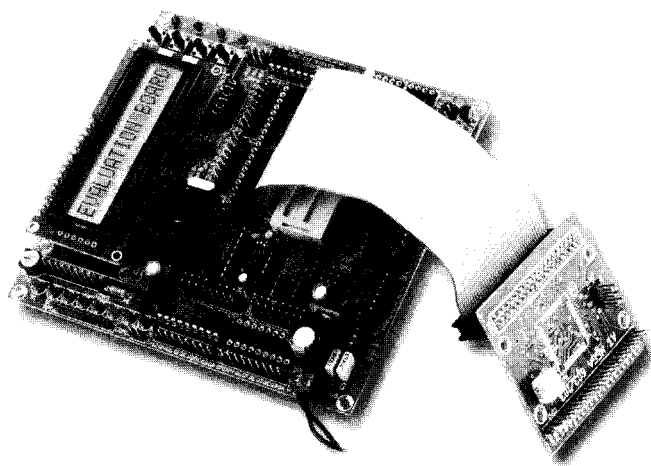
Demonstracyjną wersję Bascoma '51 można ściągnąć ze strony producenta: http://www.mcselec.com/download_8051.htm.



Rys. 9.2. Okno symulatora wyświetlaczy, który stanowi integralną część pakietu Bascom-8051

wystarczających do oprogramowania większości przeciętnych zastosowań. Olbrzymią zaletą Bascoma są dołączane przez producenta oprogramowania biblioteki obsługi m.in. magistral I²C, I-Wire, czy alfanumerycznego i graficznego wyświetlacza LCD oraz wbudowany symulator. Ciekawą cechą tego symulatora jest możliwość symulacji nie tylko zachowania samego mikrokontrolera, lecz także dołączonego do niego alfanumerycznego wyświetlacza LCD oraz 7-segmentowego (rys. 9.2).

Na uwagę zasługuje również fakt obecności na rynku wielu zestawów edukacyjnych (zestawów sprzętowych) dostosowanych do współpracy z Bascomem i znacznej liczby darmowych (np. do ściągnięcia z Internetu) opracowań konstrukcji przygotowanych za pomocą tego kompilatora. Wiele interesujących zestawów sprzętowych współpracujących z Bascomem-8051 opisano w miesięczniku Elektronika Praktyczna. Są to m.in.:



Fot. 9.3. Uniwersalny zestaw uruchomieniowy AVT-992 dla mikrokontrolerów 8051

zestaw uruchomieniowy AVT-992 (opis opublikowano w EP1/2001, zdjęcie urządzenia pokazano na fot. 9.3), programatory mikrokontrolerów AVT-887 (EP9/2000) i AVT-893 (EP10/2000).

Warto wspomnieć także o inicjatywie Sandeepa Dutta, który w roku 1999 udostępnił portalowi Source Forge (<http://www.surceforge.net>) źródła pisanego przez siebie kompilatora C dla mikrokontrolerów '51. Nosi on nazwę SDCC (*Small Device C Compiler*) i jest dostępny całkowicie bezpłatnie na prawach licencji GPL. W skład pakietu SDCC wchodzi m.in. symulator SDCDB oraz kompilator umożliwiający m.in. programowanie hybrydowe ze stosowaniem wstawek assemblerowych. SDCC jest dostępny w wersjach dla Linuxa (i systemów pochodnych) oraz Windows.

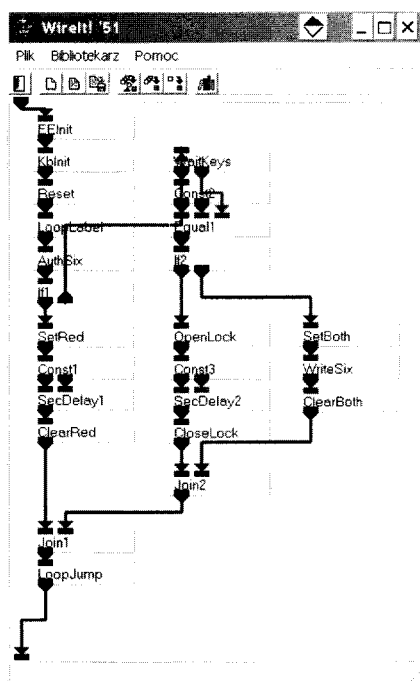


Kompilator SDCC jest dostępny (bezpłatnie) w Internecie pod adresem: <http://sdcc.sourceforge.net>.

9.3. Kompilatory niestandardowe

Jak wiadomo, od zarania dziejów każdy wynalazek zyskiwał natychmiast zarówno swoich zwolenników, jak i przeciwników. Podobnie rzecz się ma z oprogramowa-

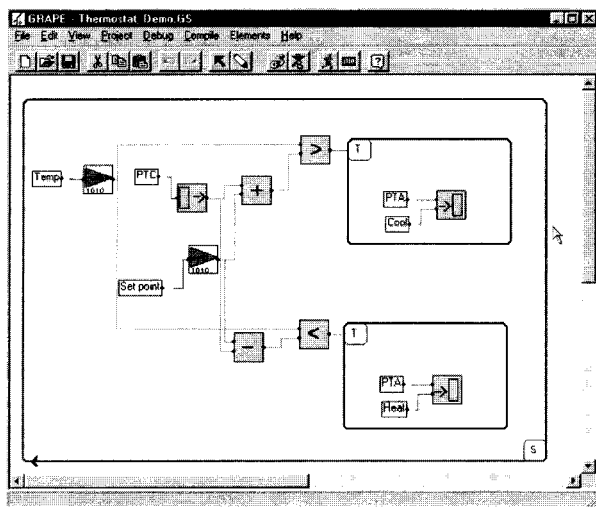
niem – praktycznie niemal od zaistnienia pierwszych potrzeb na oprogramowanie, świat podzielił się na grupkę entuzjastów i reszłę tych, którzy odczuwali chroniczny wręcz wstręt do czynności określanej mianem pisania oprogramowania. Jeśli uwzględnimy fakt, że tworzenie oprogramowania nie jest umiejętnością niezbędną do życia, trudno dziwić się temu, że producenci narzędzi uruchomieniowych przez długie lata projektowali swoje narzędzia pod kątem zastosowań co najmniej półprofesjonalnych, czyli zakładając, że ich użytkownicy muszą posiadać pewne umiejętności z zakresu programowania. Rosnąca liczba amatorów, którzy chcieliby czerpać przyjemność z tworzenia urządzeń mikroprocesorowych, ale bez konieczności zagłębiania się w dziedzinę informatyki, spowodowała jednak, że zaczęły pojawiać się na rynku rozwiązania bardziej przystępne dla szerszych kręgów osób zainteresowanych elektroniką. Przykładami takich narzędzi są WireIt oraz Grape.



Rys. 9.4. Okno edycyjne kompilatora WireIt

Pierwszym przedstawianym przykładem kompilatora graficznego opisu algorytmu przeznaczonego dla mikrokontrolerów rodziny '51 jest WireIt (opracowany przez Stanisława Skowronka). Jest to bardzo proste środowisko graficzne, umożliwiającym przedstawienie algorytmu działania programu w postaci bloczków, symbolizujących wykonywane operacje, połączonych za sobą „przewodami” sygnalizującymi kolejność wykonywania operacji (wątek, *thread*) oraz przedstawiającymi wzajemną zależność danych źródłowych i wynikowych (rys. 9.4). Każdy z bloczków ma definiowane przez użytkownika tzw. porty, określające rodzaj danych (np. liczba 8-, 16-bitowa, zmienna logiczna) oraz kierunek ich przepływu – port wyjściowy oznacza, że dane będą z bloku wysyłane jako dane źródłowe dla innego bloczku, port wejściowy, że dane są przez dany blok pobierane. Rodzaj wykonywanej operacji jest określony przez typ bloku (np. blok dodawania 16-bitowego). W programie występują predefiniowane bloki, które umożliwiają m.in. wykonywanie podstawowych operacji arytmetycznych na 8- i 16-bitowych liczbach całkowitych, operacji na zmiennych logicznych, ale także niektórych bardziej wyrafinowanych procedur (np. obsługi wyświetlacza alfanumerycznego LCD). Użytkownik ma wreszcie możliwość składania z bloczków (lub jawnej deklaracji w kodzie assemblera) własnych procedur, które następnie można dołączyć do istniejącej biblioteki, skąd można już je później pobierać tak samo jak predefiniowane bloczki. Na tym, niestety, kończą się zalety WireIt. Do wad programu należy zaliczyć m.in. brak wbudowanego assemblera (program generuje pliki typu ASM, które następnie należy kompilować do postaci binarnej lub HEX za pomocą innych narzędzi) i skromną bibliotekę predefiniowanych bloków (nie ma np. bloków operacji zmiennoprzecinkowych). Odrębną kwestią jest sam sposób obsługi programu – w szczególności konieczność ręcznego definiowania absolutnie wszystkich parametrów bloczków i portów, ograniczenie (ze względu na brak możliwości przewijania) obszaru rysowania algorytmu do pojedynczej powierzchni ekranu komputera, na którym zmieści się co najwyżej niewielka procedura oraz słaba czytelność przyjętej graficznej postaci algorytmu (wszystkie bloki wyglądają identycznie, zatem o rodzaju wykonywanej przez blok operacji można dowiedzieć się dopiero podglądając parametry bloku). Duże zastrzeżenia budzi również brak przyzwoitej instrukcji obsługi oraz polskiego interfejsu użytkownika. Wymienione niedoskonałości dość skutecznie zniechęcają do tworzenia oprogramowania za pomocą WireIt. Mimo to program pozostaje niewątpliwie ciekawym przykładem niekonwencjonalnego podejścia do problemu symbolicznego odwzorowywania algorytmu z możliwością jego kompilacji.

Zdecydowanie bardziej dopracowanym rozwiązaniem jest komercyjny (niestety!) pakiet Grape (*GRAphical Programming for Embedded systems*) firmy D.S. Grape Ltd. Pakiet ten również wykorzystuje graficzną formę przedstawiania algorytmu w postaci schematu blokowego, w którym każdy blok symbolizuje pewną operację (element funkcjonalny) systemu, zaś połączenia bloków odzwierciedlają przepływ danych (rys. 9.5). Grape jest narzędziem kompleksowym, składającym się z następujących modułów: graficzny edytor algorytmów, kompilator (przetwarzający postać graficzną algorytmu na kod pośredni), konsolidator (linker), generator kodu maszynowego, symulator mikrokontrolera oraz programowy interfejs umożliwiający współpracę pakietu z emulatorami mikrokontrolerów. W odróżnieniu od innych kompilatorów graficznego opisu algorytmów dostępnych na rynku, Grape jest programem, który generuje docelowy kod maszynowy (a nie np. kod źródłowy w języku C, który



Rys. 9.5. Okno edycyjne kompilatora Grape

należałoby potem kompilować korzystając z oddzielnego oprogramowania) i nie wymaga w związku z tym żadnej znajomości jakiegokolwiek języka wysokiego poziomu, czy asemblera. Pakiet Grape zawiera bogate biblioteki elementów standardowych – należą do nich zarówno elementy podstawowe (realizujące operacje arytmetyczne, czy logiczne), jak i elementy na wyższym poziomie abstrakcji (jak układy licznikowe, czy przetworniki A/C); można też w nim deklarować własne elementy biblioteczne. Forma opisu stosowana w pakiecie Grape umożliwia korzystanie z wszelkich standardowych konstrukcji występujących w językach wysokiego poziomu, jak np. tablice, stałe i zmienne znakowe, instrukcje typu FOR oraz WHILE.

W odróżnieniu od standardowych języków wysokiego poziomu (jak choćby język C), Grape był konstruowany od początku jako kompilator przeznaczony do tworzenia firmware'u mikrokontrolerów. W rezultacie, obsługa przerwań, sterowanie linii wejść/wyjść oraz innych typowych wewnętrznych elementów występujących w mikrokontrolerach jest w Grape wbudowanym mechanizmem standardowym, a nie doklejoną łatką, jak w przypadku klasycznych języków wysokiego poziomu (tworzonych pierwotnie z myślą o wykorzystaniu do tworzenia oprogramowania komputerowego, a dopiero później przystosowanych do tworzenia firmware'u mikrokontrolerów). Grape jest narzędziem umożliwiającym generowanie kodu maszynowego dla różnych mikrokontrolerów – dzięki temu algorytm skompilowany do kodu mikrokontrolera rodziny '51 może być łatwo przeniesiony na inny mikrokontroler tej samej rodziny, czy też na zupełnie inny typ mikrokontrolera (np. układy PIC serii 16 firmy Microchip, czy HC08 Motoroli).

Uwzględniając fakt, że efektywność wyjściowego kodu maszynowego w przypadku kompilacji w Grape jest zbliżona do rezultatów uzyskiwanych przy użyciu przyzwoitych



Demonstracyjną wersję pakietu Grape można ściągnąć ze strony producenta: <http://www.grapesys.com/download1.html>.

kompilatorów klasycznych języków wysokiego poziomu, należy stwierdzić, że Grape jest ciekawą alternatywą narzędzia do tworzenia oprogramowania dla mikrokontrolerów, zwłaszcza dla osób nie mających doświadczenia, ani nadmiernej chęci zgłębiania tajników programowania z wykorzystaniem języków wysokiego poziomu w ujęciu klasycznym.

9.4. Symulatory

Symulator mikrokontrolera jest programem umożliwiającym uruchamianie oprogramowania (*firmware'u*) przez jego wczytanie i wykonywanie na wirtualnym (symulowanym programowo na komputerze) mikrokontrolerze. Narzędzie to doskonale nadaje się do testowania procedur obliczeniowych i niedużych programów. Podstawowym problemem występującym podczas jego wykorzystywania jest utrudniona symulacja rzeczywistego świata otaczającego mikrokontroler, toteż przy użyciu symulatora trudno jest uruchamiać większe programy, intensywnie komunikujące się z zewnętrznymi urządzeniami peryferyjnymi i/lub reagującymi na różnorodne zdarzenia zewnętrzne.



Symulatory mikrokontrolerów '51 można znaleźć w Internecie m.in. pod adresami:

- <http://www.amwaw.edu.pl/~adybkows/debugger.html> - symulator wraz ze źródłami,
- <http://bit.kuas.edu.tw/~8051/> - linki do kilku symulatorów '51,
- <http://home.t-online.de/home/Jens.Altmann/jsim-e.htm> - symulator JSIM51,
- http://www.hw-server.com/x51_simulators.html - zbiór linków do bezpłatnych symulatorów.

9.5. Emulatory mikrokontrolerów

Emulator mikrokontrolera jest urządzeniem, które imituje pracę mikrokontrolera w taki sposób, że urządzenie (system) uruchamiane z wykorzystaniem emulatora zachowuje się praktycznie identycznie jak urządzenie z docelowym mikrokontrolerem (przynajmniej z punktu widzenia zachowania *firmware'u*, ponieważ mogą występować nieznaczne różnice dotyczące np. niektórych parametrów elektrycznych - choćby wartości prądu zasilania). Do niedawna emulatory mikrokontrolerów były konstruowane niemal wyłącznie na podstawie tzw. *bond-out chipów*, czyli specjalnych wersji mikrokontrolerów, które różnią się od swych zwykłych odpowiedników tym, że oprócz standardowych wyprowadzeń (jak w zwykłej wersji mikrokontrolera) mają również końcówki umożliwiające dostęp do wewnętrznej szyny adresów i danych oraz do wybranych wewnętrznych sygnałów sterujących. Emulatory są zatem narzędziami, w których efekt maksymalnego podobieństwa działania docelowego mikrokontrolera i emulatora jest osiągany głównie za pomocą rozwiązań sprzętowych.

Emulatory mikrokontrolerów są zdecydowanie najlepszymi narzędziami wspomagającymi uruchamianie systemów mikroprocesorowych, gdyż umożliwiają one uruchamianie oprogramowania w systemie (urządzeniu) docelowym, w czasie rzeczywistym, zarówno z pełną szybkością, jak i w trybie pracy krokowej, z pełną kontrolą zasobów mikrokontrolera i wykonywanego programu. Ich podstawową wadą jest cena. Dla

mikrokontrolerów rodziny '51 stawki cenowe emulatorów rozpoczynają się na poziomie tysiąca dolarów, co zwłaszcza dla amatorów stanowi barierę zaporową. Na rynku można wprowadzić znaleźć tańsze narzędzia reklamowane jako emulatory, jednak niemal zawsze są to jedynie debuggery wewnątrzukładowe.

9.6. Emulatory pamięci EPROM

W przypadku uruchamiania systemu zrealizowanego na mikrokontrolerze wykorzystującym zewnętrzną pamięć programu dość powszechną techniką była swego czasu metoda prób i błędów, w której tworzony kod źródłowy kompilowano, zapisywano w pamięci EPROM umieszczanej następnie w systemie, po czym błędy oprogramowania wykrywano na drodze obserwacji zachowania się uruchomionego urządzenia. Główną zaletą tej metody był niski koszt narzędzi uruchomieniowych, jej podstawową wadą zaś wysoka uciążliwość ciągłego przeprogramowywania pamięci EPROM. Udoskonalenie tej techniki polegało na użyciu emulatora pamięci EPROM, czyli narzędzia, które miało sondę wtykaną w podstawkę pamięci EPROM i, dzięki wewnętrznej pamięci RAM emulującej pamięć nieulotną, umożliwiało łatwą i szybką wymianę programu, bez konieczności przeprogramowywania układów EPROM. W chwili obecnej emulatory pamięci EPROM są stosowane coraz rzadziej, do czego w znacznej mierze przyczyniło się coraz częstsze wykorzystywanie mikrokontrolerów o wyłącznie wewnętrznej pamięci programu oraz szerokie rozpowszechnienie technologii pamięci nieulotnych typu Flash, które można w prosty sposób wielokrotnie wymazywać i zapisywać, także już po zamontowaniu ich w uruchamianym urządzeniu.

9.7. Monitory

Monitor jest oprogramowaniem zapisanym w pamięci programu mikrokontrolera (*firmwarem*) umożliwiającym zarządzanie uruchamianym programem. Podstawową funkcją jakiej oczekuje się od monitora jest możliwość podglądania stanu wszystkich lub wybranych rejestrów i pamięci danych. Bardziej zaawansowane monitory pozwalają na modyfikację zawartości rejestrów i pamięci, mogą też być wyposażone w bardziej zaawansowane funkcje umożliwiające np. uruchomienie pracy krokowej lub inne operacje pozwalające na implementację debuggera wewnątrzukładowego.



Kody źródłowe monitorów dla mikrokontrolerów '51 są dostępne w Internecie m.in. pod adresami:

- <http://www.pjrc.com/tech/8051/paulmon2.html>,
- http://www.hw-server.com/x51_os.html - zbiór linków do kodów źródłowych i dokumentacji różnych monitorów.

9.8. Debuggery wewnątrzukładowe

Debugger (czyt. *debager*, dosł. odpluskwiacz) wewnątrzukładowy (*in-circuit debugger*) jest rozwiązaniem pośrednim pomiędzy symulatorem i emulatorem mikrokontrolera. W odróżnieniu od symulatora, który służy do uruchamiania oprogramowania w oderwaniu od sprzętu, debugger umożliwia uruchamianie firmware'u w urządzeniu docelowym. Podobnie jak w przypadku emulatora, minimum funkcjonalności jakiej

oczekuje się od debugera, to możliwość załadowania testowanego oprogramowania do uruchamianego urządzenia oraz jego wykonywania w trybie krokowym lub z pełną szybkością, z możliwością zastawiania pułapek. Pułapki są znacznikami, które powodują, że po osiągnięciu określonego stanu (np. adresu pamięci programu) wykonywanie programu z pełną prędkością jest zatrzymywane, a użytkownik ma możliwość podglądu (i ewentualnie modyfikacji) stanu mikrokontrolera i jego zasobów a następnie kontynuacji wykonywania programu (znow w trybie krokowym lub z pełną szybkością). W odróżnieniu od emulatorów, debuggery wykorzystują zwykle rozwiązania sprzętowe w nieznacznym stopniu. Główną zaletą programowej realizacji takich funkcji jak pułapki czy praca krokowa jest zasadnicza redukcja kosztów sprzętowych debuggera (w porównaniu do emulatora), co przekłada się oczywiście na cenę narzędzia.

Podstawowa koncepcja, na której opiera się działanie debuggera, to zapisanie uruchamianego programu (*firmware*) do pamięci RAM widzianej przez mikrokontroler jako pamięć programu, a następnie wykonywanie go pod kontrolą monitora (umieszczonego w pamięci nieulotnej, również widzianej jako pamięć programu) sterowanego z zewnątrz (zazwyczaj za pośrednictwem interfejsu szeregowego RS232) za pomocą komputera klasy PC. W praktyce wystarczy, jeśli monitor będzie w stanie zrealizować następujące funkcje: odczyt i zapis wybranego rejestru lub komórki pamięci RAM (wewnętrznej lub zewnętrznej) oraz całkowite oddanie kontroli uruchamianemu programowi (wyjście z monitora). Zestaw wymienionych wyżej funkcji umożliwia realizację wszystkich istotniejszych funkcji debuggera, łącznie z implementacją pułapek i pracy krokowej.

Niestety, w odróżnieniu od emulatora, debugger wprowadza pewne ograniczenia co do uruchamianego sprzętu i oprogramowania. Najistotniejszym z nich jest konieczność użycia mikrokontrolera, który może korzystać z zewnętrznej pamięci programu. Niezbędne jest także odpowiednie sprzętowe przygotowanie uruchamianego urządzenia do współpracy z debuggerem – oprócz monitora system należy wyposażyć w pamięć RAM, która w określonym obszarze adresowym będzie widziana jako pamięć programu, ale do której mikrokontroler może również zapisywać dane jak do zwykłej pamięci RAM (przykład takiego rozwiązania pokazano na rys. 9.6). Jeśli wymiana informacji z komputerem nadrzędnym odbywa się za pomocą łącza szeregowego, to konieczne jest zajęcie przez debugger (na czas uruchamiania) układu łącza szeregowego i jednego z wewnętrznych liczników generujących sygnał taktowania (licznika określającego prędkość transmisji). Przy tym, ponieważ monitor jest zwykle implementowany jako procedura obsługi przerwania od łącza szeregowego, przerwanie to musi mieć najwyższy priorytet (żeby realizacja komend wysyłanych z komputera nadrzędnego nie była przerywana przez procedury obsługi pozostałych przerw), co w przypadku większości mikrokontrolerów rodziny '51 (a w szczególności łatwiej dostępnych układów) skutecznie uniemożliwia szersze zmiany wzajemnych priorytetów przerw (ze względu na jedynie dwa poziomy priorytetu i stosunkowo niski priorytet naturalny przerw od łącza szeregowego). Mimo wielu wymagań i ograniczeń jakie na uruchamiany system i jego oprogramowanie narzuca stosowanie debuggera wewnętrzzukładowego, jest to narzędzie często stosowane, gdyż oferowane przezeń możliwości, zwłaszcza odniesione do ceny narzędzia, są całkiem pokaźne.

9.9. Zestawy demonstracyjne, edukacyjne i prototypowe

Zestawy demonstracyjne, edukacyjne i prototypowe nie są same w sobie narzędziami uruchomieniowymi, mogą być jednak bardzo przydatne w przypadku opracowywania próbných lub częściowych rozwiązań projektów, w których autor rozwiązania chce się np. szybko zapoznać z nowym rodzajem mikrokontrolera, specjalizowanego firmware'u lub układu peryferyjnego (dla którego zaprojektowano zestaw). Podstawową zaletą zestawów demonstracyjnych, edukacyjnych i prototypowych jest fakt, iż otrzymywany zestaw jest w pełni sprawdzony pod względem działania sprzętowego, zatem przedsięwzięcie uruchamiania zostaje ograniczone głównie (lub wyłącznie) do uruchamiania firmware'u. W przypadku pakietów demonstracyjnych i edukacyjnych producent z reguły dołącza do pakietu zestaw gotowych do wykorzystania przykładowych programów i procedur, których analiza ułatwia zrozumienie sposobu działania wybranych elementów pakietu; nic też nie stoi na przeszkodzie, by procedury te wykorzystywać później w tworzonym przez siebie oprogramowaniu. Nierzadko w skład pakietu edukacyjnego lub demonstracyjnego wchodzi zestaw podstawowych narzędzi uruchomieniowych (zawierający niekiedy nawet kompilator języka wysokiego poziomu). Pakiety prototypowe znajdują zastosowanie głównie w sytuacjach, gdy zachodzi potrzeba szybkiego opracowania urządzenia, w którym oprócz standardowego, typowego dla danego mikrokontrolera rdzenia systemu ma być wykorzystywana niewielka liczba elementów zewnętrznych, a skorzystanie ze zmontowanego i wstępnie uruchomionego rdzenia systemu pozwala zaoszczędzić czas, jaki należałoby poświęcić choćby na zaprojektowanie i wykonanie płytki drukowanej.

9.10. Programatory

Programatory nie są wprawdzie narzędziami uruchomieniowymi sensu *stricto*, rozumianymi jako narzędzia umożliwiające tworzenie kodu lub wspomagające wykrywanie błędów oprogramowania, jednak w znacznej liczbie przypadków przygotowywany firmware jest ostatecznie umieszczany w wewnętrznej pamięci programu mikrokontrolera. W przypadku urządzeń małoseryjnych, w szczególności zaś przy opracowaniach modelowych i prototypowych, operacja ta jest zwykle wykonywana z zastosowaniem programatora. Upowszechnienie się technologii pamięci Flash i programowania wewnątrzukładowego (ISP od *in-system programming*) spowodowały, że obecnie do programowania znacznej liczby mikrokontrolerów coraz rzadziej są już potrzebne wyrafinowane programatory, gdyż operację przeprogramowania układu można przeprowadzić za pomocą prostej (i taniej!) przejściówki podłączanej z jednej strony do miniaturowego złącza w uruchamianym układzie, z drugiej zaś do portu szeregowego lub równoległego komputera. Znaczna liczba takich opracowań, podobnie zresztą jak i prostych programatorów w ujęciu tradycyjnym, jest publikowana w czasopiśmie



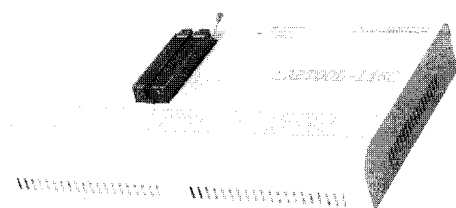
Zbiór linków do stron zawierających opisy różnych programatorów można znaleźć pod adresem:

http://www.hw-server.com/x51_programators.html.

o tematyce elektronicznej oraz w Internecie, dlatego dalej przedstawione zostaną tylko dwa proste rozwiązania programatorów mikrokontrolerów rodziny '51.

9.10.1. Programatory stacjonarne

Konstruktorzy bardzo chętnie korzystają z mikrokontrolerów z pamięcią programu typu Flash, która może być programowana w systemie (np. za pośrednictwem interfejsu szeregowego), jednak w dalszym ciągu znaczna część dostępnych na rynku mikrokontrolerów jest wyposażona w standardowe pamięci programu typu EPROM (rozdział 6), do programowania których niezbędny jest tradycyjny programator. Może to być programator uniwersalny, który umożliwia zazwyczaj programowanie nie tylko wielu typów mikrokontrolerów (różnych rodzin i producentów), ale także układów programowalnych, pamięci szeregowych i równoległych EPROM, EEPROM, Flash itp. (przykłady dwóch takich programatorów uniwersalnych pokazano na fot. 9.7 i 9.8). Programatory tego typu są rzeczywiście bardzo uniwersalne, ale też relatywnie drogie, toteż najczęściej są one wykorzystywane w firmach i instytucjach zajmujących się konstruowaniem i produkcją urządzeń elektronicznych. Przez amatorów częściej wykorzystywane są programatory przystosowane do programowania ograniczonej liczby mikrokontrolerów należących do jednej rodziny. Opisy takich programatorów przedstawiono w dalszej części tego rozdziału.



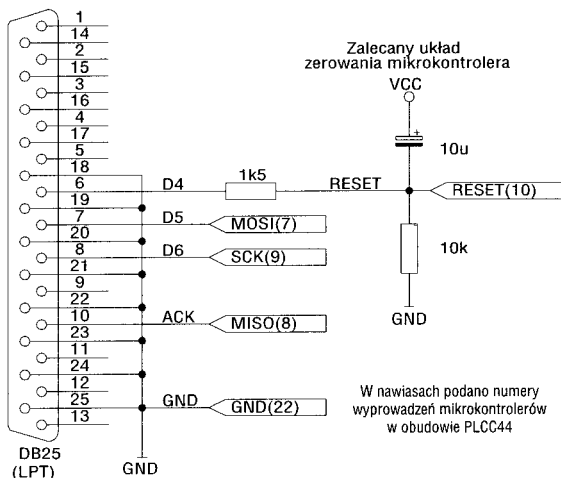
Fot. 9.7. Wygląd uniwersalnego programatora stacjonarnego Labtool-148C firmy Advantech



Fot. 9.8. Wygląd uniwersalnego programatora stacjonarnego ALL-11 firmy HiLo Systems

9.10.2. Programator wewnątrzukładowy firmy AEC Electronics

Programator firmy AEC Electronics wykorzystuje możliwość programowania mikrokontrolerów AT89S51, AT89S52, AT89S53 oraz AT89S8252 za pomocą interfejsu szeregowego SPI. Jego podstawową zaletą jest możliwość wykonywania operacji programowania wewnątrzukładowo, czyli po zamontowaniu mikrokontrolera w układzie docelowym i bez konieczności jego demontażu, czy choćby wyjmowania z podstawki w przypadku potrzeby zmiany zawartości pamięci programu. Dzięki temu ewentualna wymiana zawartości wewnętrznej pamięci programu mikrokontrolera może być łatwo przeprowadzona w dowolnym momencie, co stanowi olbrzymią zaletę np.



Rys. 9.9. Schemat programatora wewnątrzukładowego firmy AEC Electronics

w przypadku konieczności serwisowej wymiany firmware'u, związanej z wykryciem niedostrzeżonych wcześniej błędów oprogramowania. Nie bez znaczenia jest też uniezależnienie się od konieczności stosowania specjalizowanych przejściówek w przypadku stosowania tradycyjnego programatora (z programowaniem równoległym) i układów w różnych rodzajach obudów (DIP, PLCC itp.).

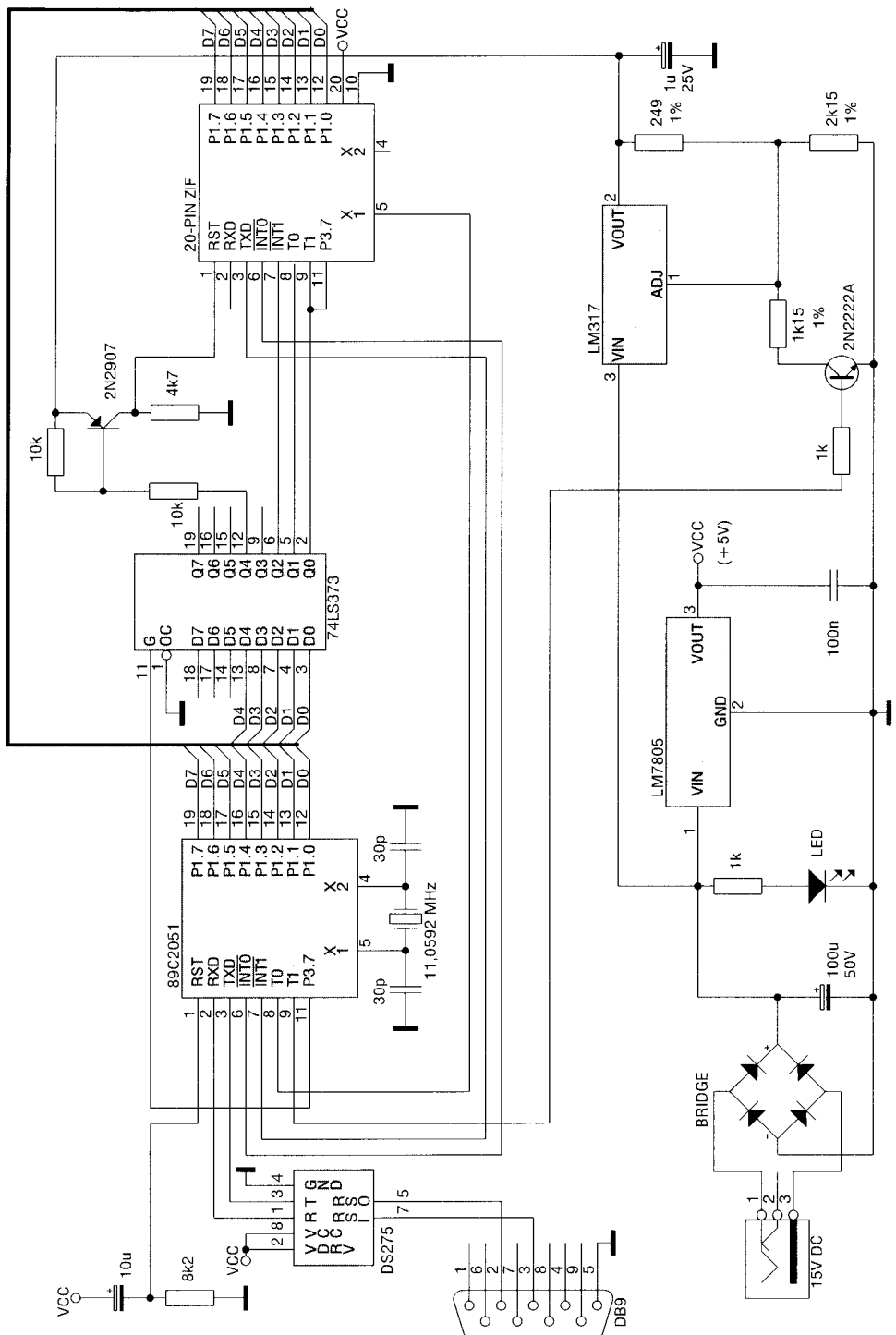
Programator korzysta z łącza drukarkowego komputera klasy PC. Sposób wykonania przejściówki ze złącza drukarkowego DB25 do układu docelowego (z mikrokontrolerem) przedstawiono na rys. 9.9. Podczas wykonywania przejściówki należy zwrócić uwagę na to, by długość kabli nie przekraczała jednego metra, gdyż ze względu na stosunkowo dużą szybkość pracy, przy większych długościach kabli mogą występować błędy transmisji. Ponieważ niektóre układy wyjść portów drukarkowych występujące w komputerach są tak skonstruowane, że w przypadku braku zasilania programowanego mikrokontrolera wypływa z nich prąd o znacznym natężeniu, warto rozważyć drobną modyfikację przejściówki polegającą na włączeniu na liniach doprowadzających sygnały do linii P1.5 (MOSI) P1.6 (MISO) i P1.7 (SCK) szeregowych rezystorów o wartości 1,5 kΩ, ograniczających wartość przepływającego prądu do bezpiecznych wartości. W razie wykorzystywania linii P1.5, P1.6 i P1.7 nie tylko do celu programowania mikrokontrolera (np. także do sterowania zewnętrznymi układami peryferyjnymi z interfejsem SPI) należy podjąć odpowiednie środki ostrożności, by sygnały występujące na tych liniach podczas operacji programowania mikrokontrolera nie spowodowały konfliktów wyjść.

Oprogramowanie sterujące pracą prezentowanego programatora jest przystosowane do pracy z DOS, można je uruchamiać także w oknie sesji DOS-owej Windows 95/98/Me.



Oprogramowanie sterujące pracą programatora AEC jest dostępne bezpłatnie pod adresem:

<http://www.aec-electronics.co.nz/software.htm>.



Rys. 9.10. Schemat programatora Easy Downloader

Podczas programowania programowane urządzenie musi zasilac mikrokontroler napięciem 5 V, mikrokontroler zaś musi być taktowany sygnałem o częstotliwości z zakresu 4...24 MHz (nie ma znaczenia, czy do tego celu wykorzystywany będzie rezonator kwarcowy, czy jakiegokolwiek inny generator podłączony do wyprowadzenia XTAL1). Oprogramowanie programatora wymaga, by plik z kodem maszynowym był typu HEX (w standardzie Intel HEX). Dla komputera PC pracującego z zegarem o częstotliwości 200 MHz czas potrzebny na zaprogramowanie i weryfikację całej pamięci programu układu 89S8252 w opisanym programatorze wynosi około pół minuty.

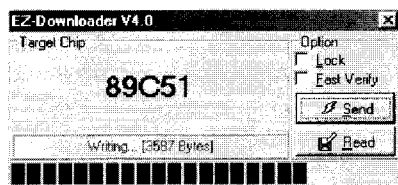
Programator firmy AEC Electronics nie umożliwia, niestety, programowania bardzo popularnych układów 89C51 i 89C52, ponieważ nie mają one wbudowanego interfejsu SPI. Z tego powodu (ze względu na dużą wygodę pracy z programatorem wewnątrzukładowym), konstruując urządzenie z mikrokontrolerami 89C51 lub 89C52 warto rozważyć możliwość zastąpienia ich na czas uruchamiania firmware'u układami 89S8252, 89S51 lub 89S52, które współpracują z opisanym programatorem.

9.10.3. Easy Downloader – prosty programator układów 89C2051 i C4051

Znaczna część niedużych konstrukcji opiera się na mikrokontrolerach 89C2051 i 89C4051. Są to niedrogie układy o niewielkiej liczbie wyprowadzeń, dlatego doskonale nadające się i chętnie wykorzystywane, do wielu miniaturowych opracowań. Układy te nie mają możliwości programowania szeregowego, jednak nawet równoległe programowanie tych mikrokontrolerów można zrealizować stosunkowo prosto – schemat

takiego programatora, opracowanego przez Wichita Sichirote'a, przedstawiono na **rys. 9.10**. Choć niewątpliwie wygodniej jest posłużyć się gotową płytką drukowaną (jej schemat montażowy oraz wzory mozaiki ścieżek znajdują się w dodatku B), to liczba połączeń na schemacie jest na tyle nieduża, że układ można wykonać nawet na płycie uniwersalnej.

Dużą zaletą programatora Easy Downloader jest to, że sterujące nim oprogramowanie EZ Downloader (dostępne bezpłatnie na stronie internetowej twórcy programatora) może pracować w dowolnym systemie Windows (także NT/2000/XP – **rys. 9.11**).



Rys. 9.11. Okno programu EZ Downloader sterującego pracą programatora Easy Downloader



Płytki drukowane do programatorów Easy Downloader w wersjach przystosowanych do programowania mikrokontrolerów '51 firmy Atmel w obudowach 20- i 40-końcówkowych są dostępne w sklepie internetowym Wydawnictwa BTC:

<http://www.btc.pl/index.php?id=15>.



Oprogramowanie sterujące pracą programatora Easy Downloader (udostępniane bezpłatnie) i szczegółowe informacje o programatorze są dostępne pod adresem:

<http://chaokhun.kmitl.ac.th/~kswichit/>.

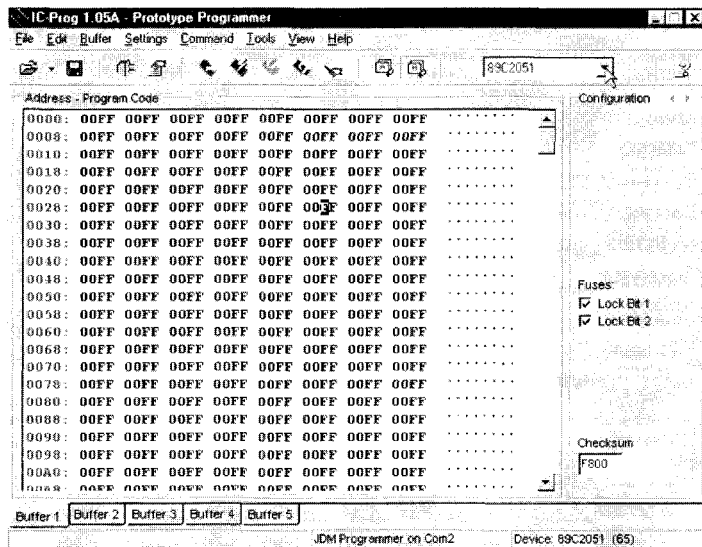
9.10.4. Oprogramowanie IC-Prog

IC-Prog (rys. 9.12) jest jednym z najpopularniejszych, bezpłatnych programów służących do obsługi programatorów ISP i standardowych programatorów równoległych. Oprogramowanie to może być wykorzystane do sterowania pracą programatora WillemProg, umożliwiając programowanie układów AT89C51/52/55, AT89LV51/52/55, AT89S8252, AT89S53, AT89LS8252, AT89LS53, AT89C1051, AT89C2051, AT89C4051, AT89C51RC, AT89C55WD, SST89C54/58, SI89C52, a także i87C51, i87C51FA, i87C51FB oraz P87LPC762, P87LPC764, P87LPC767 i P87LPC768. Zastosowanie oprogramowania IC-Prog do sterowania programatorów Tafe Programmer lub DL2TM umożliwia wprowadzić programowanie wyłącznie mikrokontrolerów AT89C2051/4051, pozwala jednak znacznie ograniczyć koszt wykonania kompletnego programatora.



Szczegółowe informacje techniczne oraz schematy programatorów współpracujących z tym oprogramowaniem znajdują się pod adresem: <http://www.ic-prog.com>.

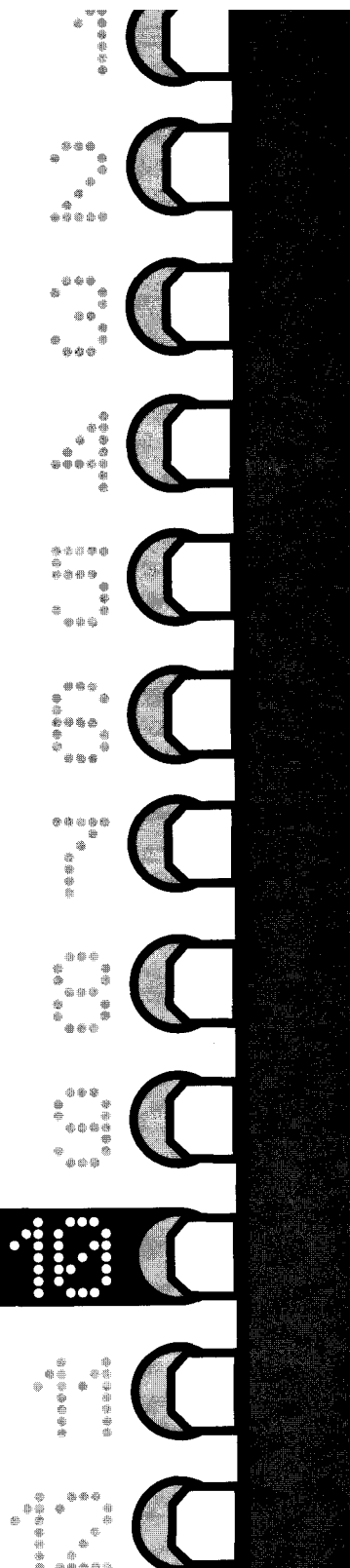
Alternatywnym rozwiązaniem sprzętowym umożliwiającym programowanie ISP mikrokontrolerów 89S53 i 89S8252 przez program IC-Prog jest programator JDM Programmer.



Rys. 9.12. Okno programu IC-Prog

Oprogramowanie IC-Prog może pracować pod dowolną wersją systemu operacyjnego Windows (łącznie z NT/2000/XP), z tym że w przypadku Windows NT i pochodnych konieczne jest zainstalowanie dodatkowego sterownika, który jest dostępny na stronie <http://www.ic-prog.com>.

**Podstawy
projektowania
urządzeń
z mikrokontrolerami
rodziny '51**



Dość powszechnym problemem wszystkich rozpoczynających swoją przygodę w krainie elektroniki jest brak odpowiedzi na podstawowe pytania: „dlaczego ktoś użył akurat takiego układu, a nie innego” lub „dlaczego tak go podłączył”? Nie da się przewidzieć wszystkich takich pytań, ani tym bardziej zgromadzić na nie wszystkie odpowiedzi. Można jedynie wybrać te najczęściej zadawane oraz te o największym znaczeniu dla poprawnego, a zarazem skutecznego projektowania urządzeń elektronicznych. Na następnych stronach znajdują się opisy najbardziej typowych elementów systemów mikroprocesorowych z przykładowymi rozwiązaniami sprzętowymi i programowymi. Przykłady zostały dobrane w taki sposób, by były w miarę elastyczne i umożliwiały składanie, nieomal jak z klocków, różnych urządzeń wykorzystujących takie same (lub bardzo podobne) elementy, tyle że w różnych konfiguracjach. Procedury towarzyszące opisom rozwiązań sprzętowych zostały również tak dobrane, by stosunkowo łatwo można je było przystosowywać do własnych potrzeb w konkretnych rozwiązaniach. Warto zwrócić szczególną uwagę na konstrukcję owych procedur i komentarze zawarte w zamieszczonych listingach. W zdecydowanej większości przypadków nie są one powieleniem tekstu książki, ale istotnym jego uzupełnieniem. Miały one na celu po pierwsze ułatwić zrozumienie działania rozwiązań sprzętowych, po drugie umożliwić oswojenie się z assemblerem mikrokontrolerów rodziny '51 na względnie prostych przykładowych procedurach. Przykłady te zostały jednak tak skonstruowane, by można w nich było przeanalizować możliwość stosowania różnych technik podejścia w programowaniu - np. przekodowywanie na podstawie tablicowanych wartości stałych lub wskutek mniej lub bardziej wyrafinowanych operacji arytmetyczno-logicznych, czytelność adresowania bezpośredniego i efektywność pośredniego. Niektóre fragmenty procedur, mimo komentarzy, będą wymagały głębszego przemyślenia, niekiedy skorzystania z szerszych opisów i przykładów wykorzystania poszczególnych instrukcji, jakie zamieszczono w rozdz. 7. Przy analizie procedur warto zwrócić uwagę na ich celowe przedstawienie jako umieszczonych w oddzielnych modułach bibliotecznych. Podejście to leży u podstaw programowania modułowego, którego główną zaletą jest to, że solidnie i względnie elastycznie opracowane klocki można później wielokrotnie wykorzystywać do tworzenia bardzo zróżnicowanych konstrukcji. Czas poświęcony na przemyślane i dokładne opracowanie modułów bibliotecznych zwraca się z nawiązką, gdyż przy kolejnych opracowaniach nie trzeba tworzyć oprogramowania od początku, ani tracić czasu na uruchamianie od zera np. n-tej wersji procedur obsługi wyświetlacza (co bywa często czynnością długotrwałą, nużącą i - zwłaszcza, jeśli powtarzaną po raz szósty - mało kształcącą). Warto zatem w miarę często stosować rozwiązania względnie uniwersalne, powtarzalne, niż optymalizować niektóre procedury (ot, choćby obsługi owego nieszczęsnego wyświetlacza) pod kątem bardzo konkretnego urządzenia, gdyż skrócenie długości kodu o jedną, czy dwie instrukcje (na kilkadziesiąt) nie będzie zwykle istotne dla prawidłowego funkcjonowania urządzenia, a może drastycznie zmniejszyć elastyczność danego rozwiązania, uniemożliwiając jego łatwe (i przyjemne) powielanie w kolejnych konstrukcjach.



Kody źródłowe programów wykorzystywanych w prezentowanych przykładach są dostępne w Internecie pod adresem:

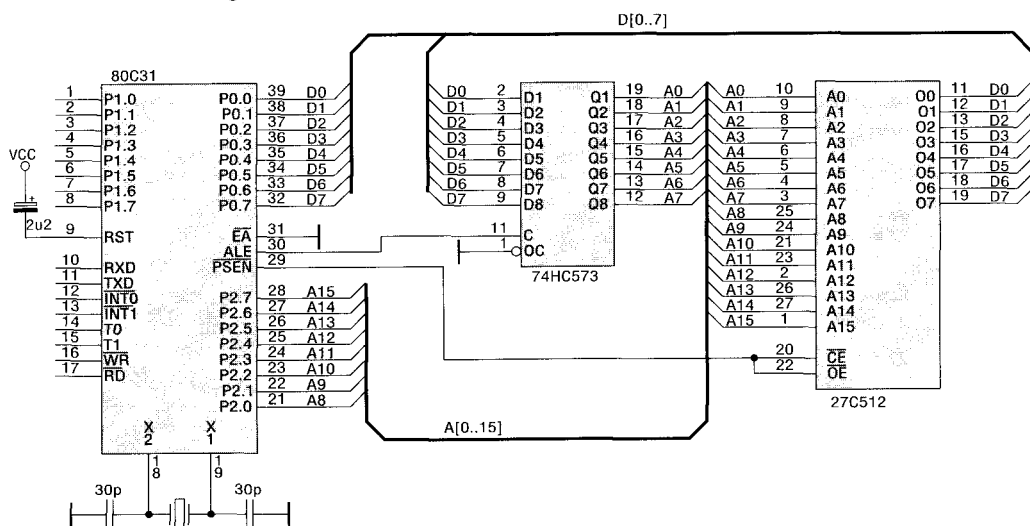
<http://www.btc.pl/index.php?id=mcs51>.

Wszystkie przykłady są napisane w assemblerze i kompilowane za pomocą kompilatora firmy IAR (<http://www.iar.com>).

10.1. Dołączenie do mikrokontrolera zewnętrznej pamięci programu

Konstruktorzy niechętnie stosują dziś rozwiązania wykorzystujące zewnętrzną pamięć programu, gdyż układy takie można bardzo łatwo skopiować (ze względu na brak jakichkolwiek zabezpieczeń pamięci programu przed odczytem), rozmiary urządzenia są większe i dwa porty mikrokontrolera (P0 i P2) trzeba poświęcić na komunikację z zewnętrzną pamięcią programu. Biorąc przy tym pod uwagę fakt, że cena mikrokontrolera z wewnętrzną pamięcią programu typu Flash jest zwykle niewiele (jeśli w ogóle) wyższa od kowoty niezbędnej na zakup elementów składowych układu z zewnętrzną pamięcią programu (tj. mikrokontrolera, zatrasku i pamięci EPROM lub EEPROM), sens stosowania zewnętrznej pamięci programu wydaje się wątpliwy. Mimo wymienionych wad stosuje się jeszcze niekiedy rozwiązania, w których pamięć programu jest układem zewnętrznym w stosunku do mikrokontrolera. Jako typowy przykład można wymienić sytuację, w której urządzenie w fazie modelowej jest uruchamiane za pomocą emulatora pamięci EPROM i/lub projektantowi nie zależy na ochronie firmware'u, gdyż jest to np. konstrukcja niekomercyjna (a na dodatek w szafie leży jeszcze zapas starych pamięci EPROM i mikrokontrolerów 80C31).

Schemat, w którym pokazano sposób dołączania zewnętrznej pamięci programu przedstawiono na rys. 10.1. Układ HC573 jest rejestrem typu *latch*, który zatraskuje opadającym zboczem sygnału ALE młodszy bajt adresu wystawiony na magistrali danych (port P0; szerszy opis sposobu pracy portów, sygnału ALE i $\overline{\text{PSEN}}$ oraz parametry czasowe – patrz rozdz. 2.5.1). Starszy bajt adresu jest pobierany bezpośrednio z wyprowadzeń portu P2. Zarówno przy stosowaniu zewnętrznej, jak i wewnętrznej pamięci programu należy bezwzględnie pamiętać o odpowiednim podłączeniu wyprowadzenia $\overline{\text{EA}}$ mikrokontrolera (patrz rozdz. 2.1.1). Należy pamiętać, że w miejsce omawianego zatrasku młodszego bajtu adresu nie można stosować układu '574,



Rys. 10.1. Sposób dołączania zewnętrznej pamięci programu do mikrokontrolerów rodziny '51

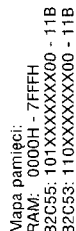
który działa wprawdzie tylko trochę, ale wystarczająco inaczej, by urządzenie przestało zupełnie funkcjonować. W starszych opracowaniach w miejscu układu '573 często można znaleźć układ '373, który jest identyczny funkcjonalnie, jednak, ze względu na sposób rozmieszczenia sygnałów na wyprowadzeniach, jest on mniej wygodny. Jeśli zatem do schematu nie został dołączony gotowy projekt płytki drukowanej, warto zastąpić go układem '573. Ze względu na pobór mocy, praktycznie wszystkie układy w technologii LS (nie mówiąc już o standardowych układach TTL) warto zastępować układami HC, HCT lub nowszymi (np. AC, AHC). Z tego samego względu stosowanie mikrokontrolerów, pamięci EPROM, czy układów peryferyjnych w technologii NMOS (czyli np. 8031 zamiast 80C31), należy uznać za naganne (nie mówiąc już o konieczności noszenia baterii do takiego urządzenia w walizce, jeśli zostało ono przypadkiem skonstruowane jako przenośne).

10.2. Współpraca mikrokontrolera z zewnętrzną pamięcią danych i zewnętrznymi układami peryferyjnymi

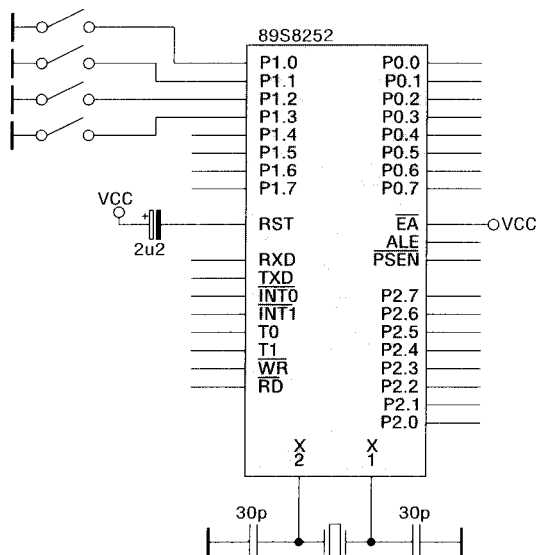
W przypadku dołączania zewnętrznej pamięci danych i/lub układów peryferyjnych sposób podejścia jest podobny do podłączania zewnętrznej pamięci programu, z tym że zamiast sygnału $\overline{\text{PSEN}}$ są używane stroby zapisu i odczytu WR i RD (funkcje alternatywne linii P3.6 i P3.7 portu P3). Należy też pamiętać o tym, że w mikrokontrolerach rodziny '51 nie występuje rozróżnienie między przestrzenią zewnętrznego pamięci danych i zewnętrznymi układów peryferyjnych (patrz rozdz. 2.1), a zatem w razie wykorzystania kilku układów pełniących funkcje zewnętrznej pamięci danych i/lub zewnętrznymi układów peryferyjnych, konieczne będzie dekodowanie adresów, by nie występowały konflikty związane z próbą jednoczesnego dostępu do kilku z tych układów. W obydwu przypadkach (tj. pamięci i układów peryferyjnych) do sterowania zapisem i odczytem danych służą instrukcje typu MOVX, wykorzystujące adresowanie 8- lub 16-bitowe (patrz rozdz. 2.5.1 i opis instrukcji MOVX w rozdz. 7). Schemat prezentujący przykładowy sposób dołączenia zewnętrznej pamięci danych i układów peryferyjnych przedstawiono na rys. 10.2.

10.3. Obsługa klawiatur

Znaczna część urządzeń budowanych z wykorzystaniem mikrokontrolerów jednoukładowych jest wyposażona w mniej lub bardziej rozbudowane klawiatury, umożliwiające użytkownikowi zmianę nastaw urządzenia. W prostszym przypadku (np. w zegarku) klawiaturę stanowi zaledwie kilka klawiszy, które można podłączyć bezpośrednio do linii portów mikrokontrolera. Chcąc podłączyć klawiaturę do linii portów należy uwzględnić fakt, że w układach rodziny '51 typowa linia pracująca jako wejście jest utrzymywana w stanie wysokim przez słaby wewnętrzny rezystor podciągający (patrz rozdz. 2.4). Tak więc odczyt niewysterowanej z zewnątrz linii portu pracującej jako wejście daje wynik w postaci jedynki logicznej, a dopiero po wymuszeniu z zewnątrz stanu niskiego (w szczególności zwarcie do masy) odczyt linii daje zero logiczne.



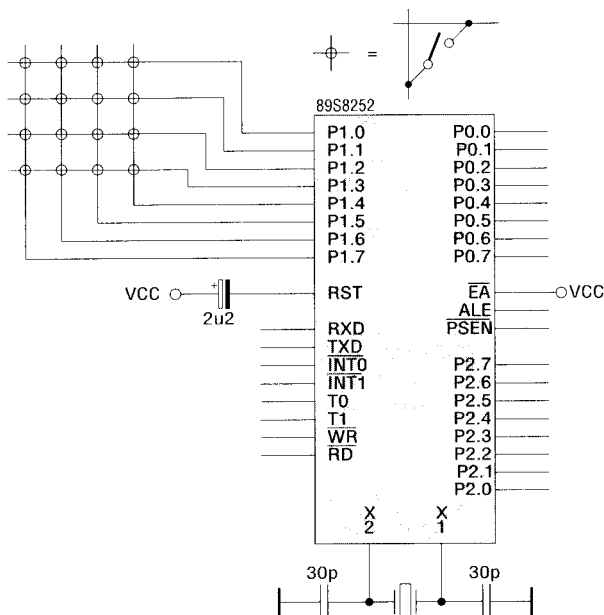
Rys. 10.2. Przykładowy schemat systemu z zewnętrzną pamięcią danych i dwoma zewnętrznymi układami peryferyjnymi (dekodowanie adresów za pomocą bramek)



Rys. 10.3. Przykładowy sposób podłączenia do mikrokontrolera klawiatury w układzie zwarcia do masy

Z tego względu klawisze umieszcza się między wejściami mikrokontrolera a masą zasilania (rys. 10.3).

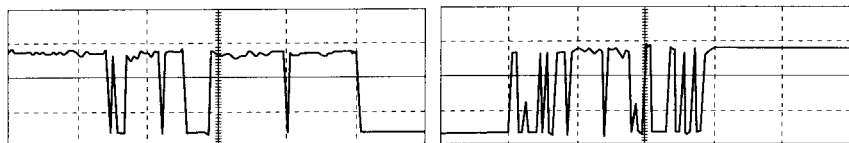
Jeśli zachodzi potrzeba zastosowania klawiatury o większej liczbie klawiszy, najczęściej wykonuje się ją w połączeniu matrycowym, w którym klawisze umieszczone są na przecięciu umownych rzędów i kolumn. Jeśli jednak liczba klawiszy nie przekracza pięciu, stosowanie matrycowego układu połączeń nie ma sensu, ponieważ nie spowoduje oszczędności w liczbie linii mikrokontrolera potrzebnych do dekodowania stanu klawiatury. W przypadku konfiguracji matrycowej również należy uwzględnić obecność wewnętrznych rezystorów podciągających, dlatego też techniką stosowaną dla zdekodowania wciśniętego klawisza jest tzw. przemiatanie zerem, czyli wymuszanie niskiego stanu na tylko jednej z kolumn (przy założeniu, że kolumny pracują jako wyjścia, a rzędy jako wejścia) i sprawdzenie stanu wszystkich rzędów. Jeśli żaden z klawiszy nie jest wciśnięty, wszystkie linie rzędów będą w stanie wysokim, jeśli natomiast odczyt któregośkolwiek z rzędów da zero logiczne, będzie to oznaczało, że klawisz umieszczony na przecięciu tego właśnie rzędu i aktywnej kolumny (czyli tej, na której wymuszono stan niski) jest wciśnięty. W ten sposób, przemiatając niskim stanem kolejno po wszystkich kolumnach i sprawdzając każdorazowo stan rzędów, można stwierdzić, czy jakiś klawisz został wciśnięty i określić jego położenie. Warto zwrócić uwagę na fakt, iż w mikrokontrolerach rodziny '51 specyficzna konstrukcja wewnętrzna stopni końcowych linii portów powoduje, że stan wysoki (zarówno podczas pracy linii jako wyjście, jak i jako wejście) jest wytwarzany przez słaby rezystor podciągający (o niskiej wydajności prądowej), a tylko w stanie niskim wydajność prądowa wyjścia jest większa (patrz rozdz. 2.4). Dzięki temu, jednoczesne zwarcie dowolnych kilku (czy nawet wszystkich) klawiszy nie spowoduje uszkodzenia układu. Przykład pokazujący sposób podłączenia klawiatury w układzie matrycowym 4×4 przedstawiono na rys. 10.4.



Rys. 10.4. Przykładowy sposób dołączenia do mikrokontrolera klawiatury w układzie matrycowym

Programowa obsługa klawiatury (zwłaszcza w układzie rys. 10.3) nie jest rzeczą skomplikowaną, jednak by działała ona poprawnie należy uwzględnić zjawisko drgania zestyków, typowe dla niemal wszystkich styków mechanicznych. Polega ono na tym, że każdorazowe wciśnięcie i puszczenie klawisza nie przebiega w sposób natychmiastowy, lecz charakteryzuje się pewnym stanem przejściowym, trwającym typowo od kilku do kilkunastu milisekund, podczas którego styki drgają, na przemian zwierając i rozwierając wyprowadzenia klawisza (rys. 10.5). Najprostszy sposób eliminacji problemu drgania zestyków w procedurze obsługi klawiatury polega na odczekaniu co najmniej 30 ms od momentu wykrycia zmiany stanu klawisza (czyli do chwili osiągnięcia przez klawisz stanu ustalonego) przed próbą ponownego odczytania stanu klawiatury. Skutkiem nieuwzględnienia w procedurze obsługi klawiatury drgań zestyków może być wielokrotne wczytanie tego samego klawisza przy jego pojedynczym naciśnięciu i/lub puszczeniu.

Wprawdzie, zwłaszcza w przypadku małej liczby klawiszy, obsługę klawiatury można zrealizować opierając się na przerwaniach zewnętrznych (podłączając klawisze do linii tych przerwań), jednak w praktyce klawiaturę podłącza się najczęściej do zwykłych linii wejść/wyjść, zaś obsługę realizuje się w procedurze przerwania od wewnętrznego



Rys.10.5. Przykład sygnału obserwowanego na wejściu mikrokontrolera, jaki powstaje w wyniku pojedynczego wciśnięcia przycisku dołączonego między to wejście i masę

układu licznikowego mikrokontrolera ustawionego w tryb odmierzenia czasu (wytworzącego przerwanie z określoną częstotliwością). Przykładową procedurę obsługi klawiatury z rys. 10.4 przedstawiono w list. 10.1.

List. 10.1. Przykładowa procedura obsługi klawiatury matrycowej pokazanej na rys. 10.4

```
; =====
; obsługa klawiatury matrycowej 4 x 4
; udostępniane są procedury jak w deklaracji PUBLIC
; wymagane zdefiniowanie sposobu podłączenia
; i dodatkowych parametrów (typowo w module głównym)
; jak w EXTERN
; niszczy A, B, PSW (w tym CY)
; =====
EXTERN port_klawiatury, klawisz, wcisniety, puszczoney
EXTERN licznik_klawiatury, kb_tick
PUBLIC klawiatura

klawiatura:
    DJNZ licznik_klawiatury, wracaj ; żeby obsługa była dla
    MOV licznik_klawiatury, #kb_tick ; f = f(T0_int)/kb_tick
    MOV port_klawiatury, #07FH ; wymuszenie zera na kolumnie
    MOV A, port_klawiatury ; odczytanie stanu rzędów
    ANL A, #0FH ; maskowanie starszej połówki
    MOV B, #0 ; częściowy kod klawisza do B
    CJNE A, #0FH, klawisz_wcisniety ; sprawdzanie, czy coś wciśnięte
    MOV port_klawiatury, #0BFH ; analogicznie na innych
    ; kolumnach

    MOV A, port_klawiatury
    ANL A, #0FH
    MOV B, #4
    CJNE A, #0FH, klawisz_wcisniety
    MOV port_klawiatury, #0DFH
    MOV A, port_klawiatury
    ANL A, #0FH
    MOV B, #8
    CJNE A, #0FH, klawisz_wcisniety
    MOV port_klawiatury, #0EFH
    MOV A, port_klawiatury
    ANL A, #0FH
    MOV B, #12
    CJNE A, #0FH, klawisz_wcisniety
    JBC wcisniety, klawisz_puszczoney ; nic nie jest wciśnięte
wracaj:
    RET

klawisz_puszczoney:
    SETB puszczoney ; sygnalizacja puszczenia
    ; klawisza
    RET ; jeśli poprzednio był wciśnięty

klawisz_wcisniety:
    SETB wcisniety ; sygnalizacja wciśnięcia
    ; klawisza

    ; teraz określany będzie numer wciśniętego klawisza
    ; częściowy numer klawisza (dwa mniej znaczące bity) jest już w B

    XCH A, B
    JNB B.3, zapisz_kod_klawisza ; a częściowo brutalnie
    ; sprawdzając
```

```

JNB B.2, dodaj_1                ; bity odczytane wcześniej
                                ; z portu
JNB B.1, dodaj_2                ; klawiatury
ADD A, #3
MOV klawisz, A                  ; kod wciśniętego klawisza do
                                ; bufora

RET
dodaj_2:
INC A
dodaj_1:
INC A
zapisz_kod_klawisza:
MOV klawisz, A                  ; kod wciśniętego klawisza do
                                ; bufora

RET
END                              ; koniec modułu

```

10.4. Sterowanie wyświetlacza i diod LED

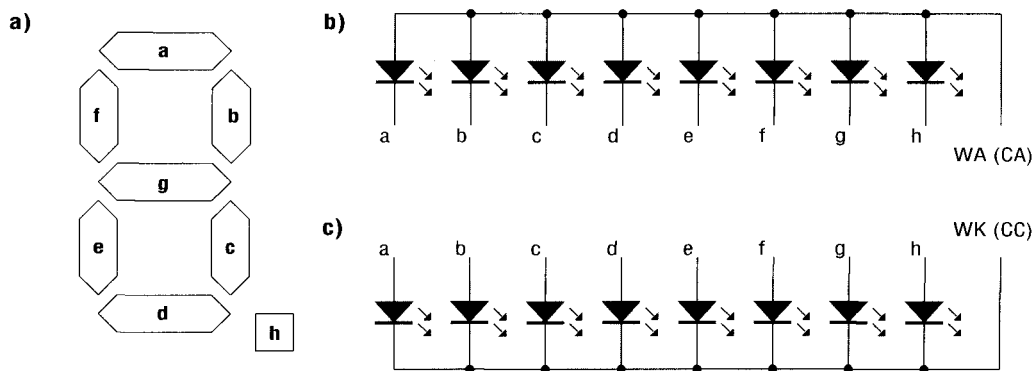
Jednym ze standardowych elementów stosowanych w układach realizowanych na mikrokontrolerach jednoukładowych jest wyświetlacz LED i diody LED. Wprawdzie wygodniejszy w użyciu jest alfanumeryczny wyświetlacz LCD, jednak niewątpliwą zaletą wyświetlacza LED, istotną zwłaszcza w zastosowaniach przemysłowych, jest jego lepsza czytelność związana zarówno z intensywnością świecenia, maksymalnym kątem dobrej widoczności, jak i możliwym do uzyskania rozmiarem znaków. Obecnie produkowane wyświetlacze LED są wykonywane jako wskaźniki 7-segmentowe, 16-segmentowe lub matrycowe. Koncepcja sterowania wszystkich tych elementów, podobnie jak i pojedynczych diod LED, jest identyczna, tak więc dalej ograniczono się do opisu układu ze wskaźnikami 7-segmentowymi, które są wykorzystywane najczęściej.

Aby „zmusić” diodę LED do świecenia, należy przepuścić przez nią prąd o odpowiednim natężeniu. W praktyce, w zależności od typu diody, jest to najczęściej prąd o wartości od kilku do kilkunastu miliamperów. Subiektywna (odczuwana przez patrzącego) jasność świecenia jest (w szerokim zakresie) proporcjonalna do średniej (!) wartości prądu płynącego przez diodę. Przepływowi prądu przez diodę LED odpowiada spadek napięcia na diodzie (zależny w znacznym stopniu od koloru światła emitowanego przez diodę), wynoszący typowo 1,8...2,3 V (dla diod świecących na czerwono, zielono oraz żółto), ale w niektórych przypadkach mogący dochodzić nawet do około 5,0 V (dla diod świecących na niebiesko). Sterowanie wskaźnikami nie różni się praktycznie niczym od sterowania zwykłymi diodami LED, gdyż wskaźnik jest niczym innym jak zespołem diod LED uformowanych w kształt poszczególnych segmentów, połączonych (w celu zmniejszenia liczby wyprowadzeń) anodami – tzw. wskaźniki ze wspólną anodą – lub katodami – tzw. wskaźniki ze wspólną katodą (rys. 10.6). Należy jednak zwrócić uwagę na to, że w większych wskaźnikach pojedyncze segmenty są wykonywane z kilku – połączonych szeregowo – diod LED, co oczywiście powoduje istotny wzrost napięcia wymaganego do wysterowania segmentów takiego wskaźnika.

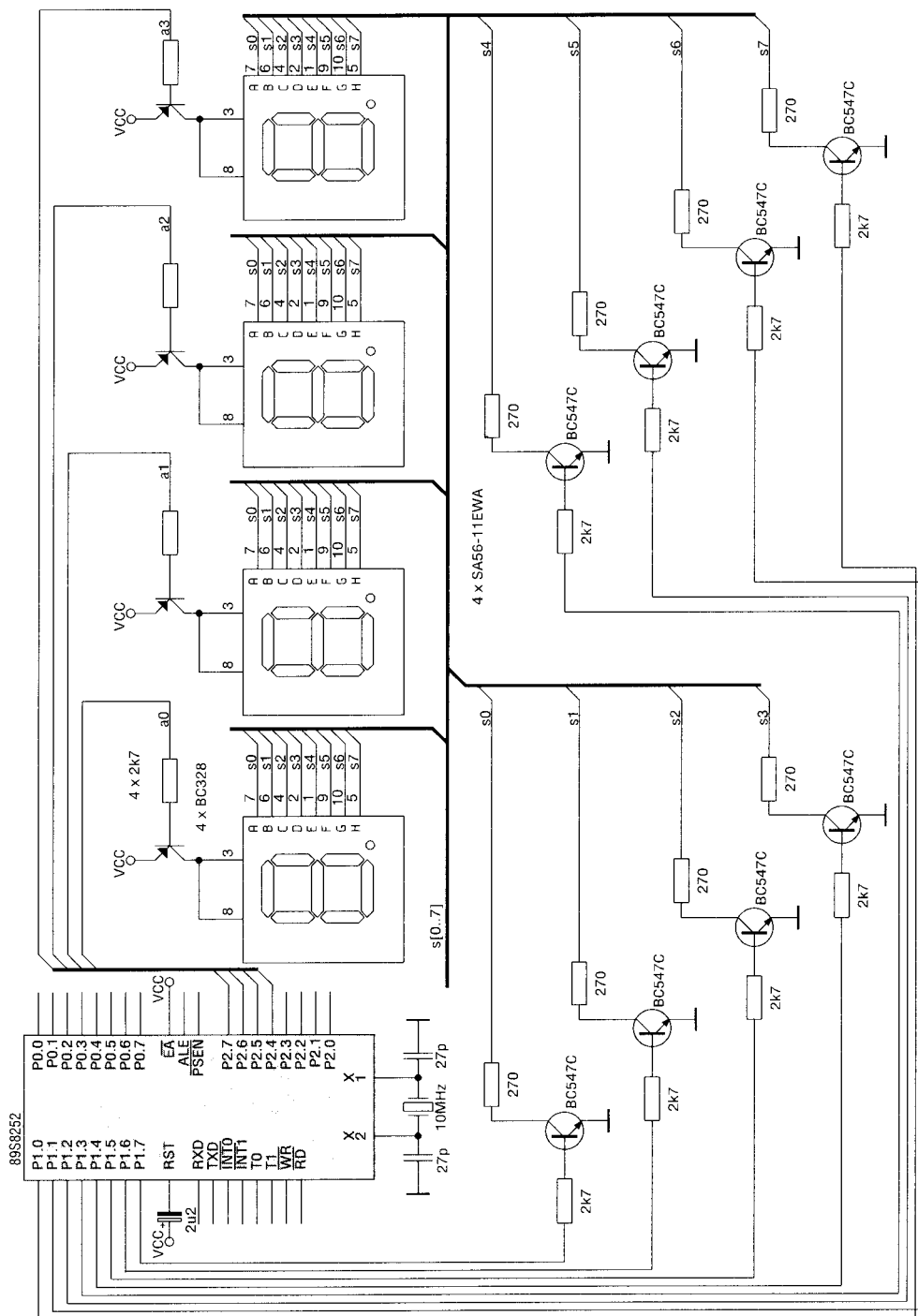
Początkowo (przed okresem intensywnego rozwoju techniki mikroprocesorowej) wyświetlacze LED były sterowane niemal wyłącznie w sposób statyczny, w którym każdy segment każdego wskaźnika był indywidualnie włączany bądź wyłączany. Jak jednak

łatwo policzyć, do sterowania statycznego 4-cyfrowego wyświetlacza 7-segmentowego (zawierającego po 8 diod LED na wskaźnik) potrzeba aż 32 linii sterujących. Taka rozrzutność jest praktycznie nie do przyjęcia w układach realizowanych na mikrokontrolerach, w których do dyspozycji jest bardzo często zaledwie kilkanaście linii portów. Dlatego też w urządzeniach mikroprocesorowych stosuje się niemal wyłącznie dynamiczne sterowanie wyświetlaczy LED. Takie sterowanie wykorzystuje bezwładność ludzkiego oka, które obraz powtarzany z częstotliwością kilkudziesięciu herców postrzega jako statyczny. Ogólna koncepcja dynamicznego sterowania wyświetlaczem LED polega zatem na tym, by połączyć ze sobą równolegle odpowiednie segmenty kolejnych wskaźników (ewentualnie także zespołów diod LED pogrupowanych w wirtualne wskaźniki) i mając możliwość oddzielnego sterowania poszczególnymi wspólnymi anodami (katodami), w danym momencie zapalać odpowiednie segmenty tylko jednego wskaźnika (załączając w tym czasie tylko jedną wspólną anodę/katodę), a po upływie np. 1 ms odpowiednie (tzn. niekoniecznie te same) segmenty kolejnego wskaźnika itd. Powtarzając cyklicznie z odpowiednią częstotliwością (np. około 100 Hz) włączanie tych samych kombinacji segmentów odpowiednich wskaźników uzyskuje się obraz postrzegany jako statyczny. Ze względu na konieczność regularnego odświeżania wyświetlacza, obsługę dynamicznie sterowanego wyświetlacza LED najczęściej realizuje się w procedurze przerwania od wewnętrznego układu licznikowego mikrokontrolera ustawionego w tryb odmierzania czasu (generującego regularne przerwanie z określoną częstotliwością).

Praktyczna implementacja procedur obsługi wyświetlaczy LED sterowanych dynamicznie nie nastręcza zwykle większych problemów (przykładową procedurę obsługi wyświetlacza przedstawionego na rys. 10.7 zamieszczono w list. 10.2). Początkującym elektronikom większe kłopoty może sprawić projekt rozwiązania sprzętowego, w którym należy odpowiednio dobrać tranzystory i rezystory ograniczające prąd segmentów. Konstrukcję układu należy rozpocząć od określenia prądu segmentów, pamiętając o tym, że jasność świecenia jest proporcjonalna do średniej wartości prądu płynącego przez segment wskaźnika, tak więc chcąc np. dla 8-cyfrowego wyświetlacza uzyskać średni prąd segmentu wynoszący 10 mA, prąd chwilowy musi wynosić 80 mA. Jeśli jednak, np. podczas uruchamiania urządzenia, przemiatanie wskaźników zostanie



Rys. 10.6. Rozmieszczenie segmentów (a) oraz sposób połączenia diod LED we wskaźniku 7-segmentowym ze wspólną anodą (b) i katodą (c)



Rys. 10.7. Przykład realizacji wyświetlacza LED sterowanego dynamicznie przez mikrokontroler

choćby chwilowo wstrzymane (przy załączonym wskaźniku), to aktywny w danym momencie wskaźnik ulegnie uszkodzeniu (bo typowy maksymalny statyczny prąd segmentu wynosi około 20 mA). Warto zatem (przynajmniej na czas uruchamiania) wartość prądu segmentów dobierać w taki sposób, by nawet przypadkowe zatrzymanie przemiatań wskaźników nie spowodowało uszkodzeń, ale by jednocześnie wartość tego prądu była wystarczająca do zaświecenia wskaźników. Tranzystory sterujące segmentami należy dobrać tak, by ich maksymalny statyczny prąd kolektora był nie mniejszy od obliczonej chwilowej wartości prądu segmentów (w podanym przykładzie 80 mA). Tranzystory sterujące wspólnymi anodami/katodami należy dobrać tak, by ich maksymalny statyczny prąd kolektora był nie mniejszy od sumy chwilowej wartości prądu segmentów obliczonej dla wszystkich segmentów podłączonych jednocześnie do danej anody/katody (w podanym przykładzie będzie to $80 \text{ mA} \cdot 8$ segmentów = 640 mA). Na końcu, znając maksymalne napięcie zasilania układu, napięcia nasycenia tranzystorów i spadek napięcia na segmentach wskaźników, wylicza się wartości rezystorów ograniczających prąd segmentów.

List. 10.2. Procedura obsługi wyświetlacza przedstawionego na rys. 10.7

```

;=====
;   przykładowa procedura obsługi wyświetlacza LED
;   sterowanego dynamicznie
;   udostępniane są procedury jak w deklaracji PUBLIC
;   wymagane zdefiniowanie kilku parametrów
;   (typowo w module głównym) - jak w EXTERN
;   niszczy ACC, R0, PSW (w tym CY)
;=====

EXTERN liczba_LEDow, nr_LED, anody, segmenty
PUBLIC wyswietlacz_LED, cyfra_na_7_seg

wyswietlacz_LED:
    ORL anody, #0F0H          ; wyłączenie anod (ustawiając cztery
                                ; starsze bity portu)

    DJNZ nr_LED, displ
    MOV nr_LED, #liczba_LEDow

displ:
    MOV A, #nr_LED            ; do adresu początku bufora wyświetlacza
    ADD A, nr_LED              ; dodaje numer aktywnego wskaźnika, co daje
    MOV R0, A                 ; adres aktywnego wskaźnika w buforze
                                ; wyświetlacza
    MOV A, @R0                ; zawartość aktywnego bajtu bufora
                                ; wyświetlacza do ACC
    MOV segmenty, A           ; a następnie do rejestru segmentów
    MOV ACC, #2                ; offset do tabeli anod
    ADD A, nr_LED              ; uwzględniający numer aktywnego
                                ; wyświetlacza
    MOVC A, @A+PC              ; wyłuskanie kodu anody z tabeli anod
    XRL anody, A               ; włączenie nowej anody
    RET

tabela_anod:
    DB 80H, 40H, 20H, 10H

;=====
;   procedura pomocnicza konwersji cyfry hex na kod 7-segmentowy
;   łatwa w przystosowaniu do dowolnego rozmieszczenia segmentów
;   i dowolnego rodzaju wskaźników (ze wspólną anodą/katodą)
;   cyfra do zdekodowania i zwrotny kod w ACC
;=====

```

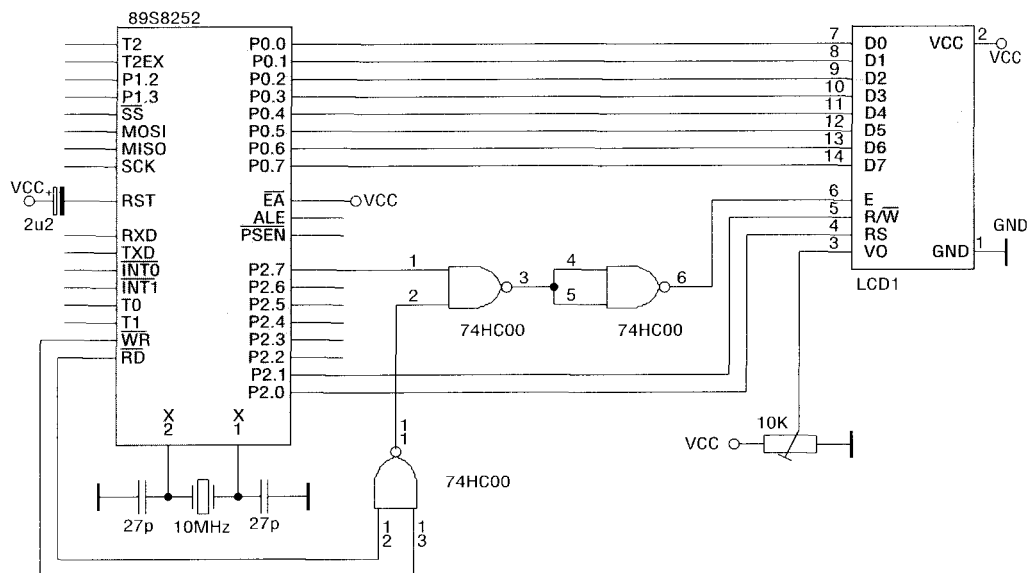
```

cyfra_na_7_seg:
    INC A                ; uwzględnienie przeskoku przez instrukcję
                        ; RET
    MOVC A, @A+PC        ; wyłuskanie kodu z poniższej tabeli kodów
    RET
    DB 11000000B        ; 0
    DB 11111001B        ; 1
    DB 10100100B        ; 2
    DB 10110000B        ; 3
    DB 10011001B        ; 4
    DB 10010010B        ; 5
    DB 10000010B        ; 6
    DB 11111000B        ; 7
    DB 10000000B        ; 8
    DB 10010000B        ; 9
    DB 10001000B        ; A
    DB 10000011B        ; b
    DB 11000110B        ; C
    DB 10100001B        ; d
    DB 10000110B        ; E
    DB 10001110B        ; F
    DB 11111111B        ; wygaszony (16)
    DB 10111111B        ; - (17)
    DB 11111110B        ; . (18)
END                      ; koniec modułu

```

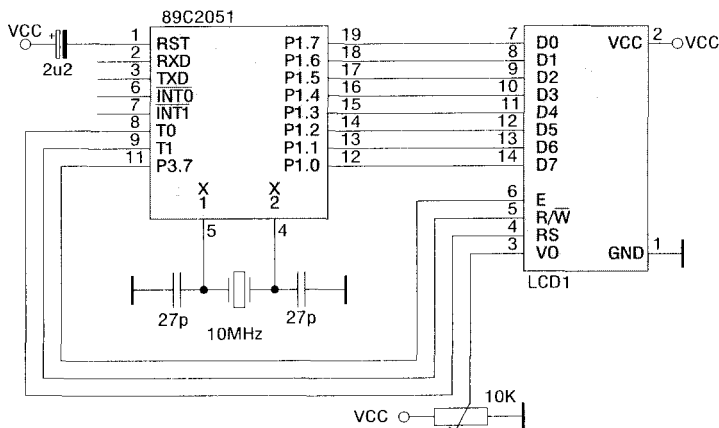
10.5. Współpraca mikrokontrolera z alfanumerycznym wyświetlaczem LCD

Wprawdzie w wielu stosunkowo prostych zastosowaniach, zwłaszcza automatyki przemysłowej, wyświetlacze LED stanowią najczęstszy wybór, to jednak w urządzeniach zasilanych bateryjnie lub takich, które wymagają nieco szerszej komunikacji tekstowej z użytkownikiem, zdecydowany prym wiodą alfanumeryczne wyświetlacze LCD. W systemach mikroprocesorowych szczególnie chętnie wykorzystywane są alfanumeryczne wyświetlacze LCD, które są wyposażone w standardowy interfejs równoległy i dzięki temu mogą być traktowane praktycznie jak klasyczne zewnętrzne układy peryferyjne. Wyświetlacze te są przystosowane do zasilania napięciem +5 V, a ich prąd zasilania wynosi zaledwie około 0,5 mA. Oprócz 8-bitowej dwukierunkowej magistrali danych (linie D0...7), która może pracować także w trybie 4-bitowym, wyświetlacze te mają tylko trzy linie sterujące. Oprócz wymienionych sygnałów i zasilania, do poprawnej pracy potrzebują już tylko napięcia sterującego poziomem kontrastu (w zdecydowanej większości przypadków z zakresu 0...5 V, choć trafiają się też modele wymagające ujemnego napięcia kontrastu). Wyświetlacze te mają wbudowany generator znaków, toteż ich sterowanie jest stosunkowo proste, gdyż podstawowe znaki kodowane są w powszechnie znanym standardzie ASCII. Nieco gorzej ma się sprawa z polskimi, czy zachodnioeuropejskimi znakami diakrytycznymi, ponieważ większość pamięci generatora znaków, o kodach powyżej 128, zajmuje japońska katakana (u nas jakoś mniej popularna niż zwykle „ą”, „ś”, czy choćby „ü”). Na szczęście, użytkownik ma możliwość zdefiniowania kilku własnych znaków, co z grubsza rozwiązuje problem polskich „ogonków”.



Rys. 10.8. Przykładowy sposób podłączenia alfanumerycznego wyświetlacza LCD jako urządzenia widzanego w przestrzeni adresowej zewnętrznej pamięci RAM mikrokontrolera

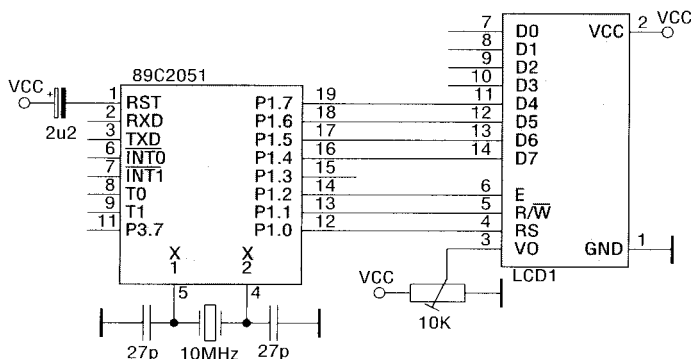
Przy podłączaniu wyświetlacza jak standardowego zewnętrznego układu peryferyjnego (sterowanego instrukcjami MOVX) należy zwrócić uwagę na dwie istotne kwestie. Sygnał strobowy E, sterujący dostępem do układu wyświetlacza, musi mieć polaryzację dodatnią, czyli odwrotną niż sygnały strobu zapisu i odczytu generowane przez mikrokontrolery rodziny '51. Stąd konieczność zastosowania przynajmniej inwertera (w przykładzie na rys. 10.8 zastosowano bramki NAND realizujące, oprócz odwracania polaryzacji strobów zapisu i odczytu, także funkcję częściowego dekodowania adresu). Drugim problemem jest stosunkowo wolny cykl dostępu do układu, który



Rys. 10.9. Przykładowy sposób podłączenia alfanumerycznego wyświetlacza LCD pracującego w trybie komunikacji 8-bitowej, z programowym wytwarzaniem sygnałów sterujących przez mikrokontroler

powoduje, że już przy częstotliwościach taktowania mikrokontrolera niewiele przekraczających 10 MHz mogą się pojawiać problemy w komunikacji z wyświetlaczem. Wprawdzie można znaleźć modele wyświetlaczy z krótszymi czasami dostępu, jednak nie stanowią one większości modeli osiągalnych na rynku.

Wymienione czynniki powodują, że jeśli wyświetlacz ma być jedynym urządzeniem korzystającym z portu P0 pracującego w trybie magistrali danych, przyłącza się go do dowolnych linii mikrokontrolera (rys. 10.9), implementując komunikację z wyświetlaczem całkowicie programowo. W przypadku niewielu linii jakie można wykorzystać do sterowania (np. w mikrokontrolerach o zredukowanej liczbie wyprowadzeń) warto zastanowić się nad możliwością wykorzystania interfejsu wyświetlacza w 4-bitowym trybie pracy (rys. 10.10). Magistralę danych stanowią wówczas jedynie linie D4...7 (linie D0...3 wyświetlacza nie są wtedy wykorzystywane), a każda operacja wymiany informacji z układem wyświetlacza wymaga podwójnego strobu na linii E (podczas pierwszego strobu jest przesyłana bardziej znacząca połówka bajtu informacji, podczas drugiego – mniej znacząca).



Rys. 10.10. Przykładowy sposób podłączenia alfanumerycznego wyświetlacza LCD pracującego w trybie komunikacji 4-bitowej, z programowym wytwarzaniem sygnałów sterujących przez mikrokontroler

Sterowanie alfanumerycznym wyświetlaczem LCD odbywa się na zasadzie przesyłania do wyświetlacza prostych komend oraz zapisu i odczytu danych. Sterownik wyświetlacza rozróżnia komendy i dane na podstawie stanu linii RS (RS = 0 oznacza wysłanie komendy, RS = 1 oznacza przesłanie danych), zaś kierunek transmisji danych (odczyt/zapis) zależy od stanu linii R/W. Przy interpretacji i wyborze poszczególnych komend istotne jest jeszcze rozróżnienie między dwoma rodzajami pamięci, jakie występują w wyświetlaczu. Pierwszym z nich jest pamięć generatora znaków, który oprócz graficznej postaci większości znaków przechowywanej w pamięci nieulotnej daje użytkownikowi możliwość zdefiniowania postaci graficznej własnych ośmiu znaków (wywołując je później do wyświetlenia należy używać kodów z zakresu



Rozmieszczenie wyprowadzeń wyświetlaczy LCD ze sterownikiem HD44780 i jego odpowiednikami pokazano w dodatku C.

od 0 do 7, identycznych z tymi jakie wykorzystano podczas definicji znaków użytkownika komendą *CG RAM set*).

Drugim rodzajem pamięci wewnętrznej wyświetlacza jest pamięć, w której przechowywane są kody (quasi-ASCII) znaków przeznaczonych do wyświetlenia (tzw. pamięć DD RAM). Pamięć DD RAM ma pojemność 80 bajtów. Jeśli wyświetlacz nie wykorzystuje wszystkich 80 bajtów pamięci DD RAM (bo jest to np. wyświetlacz 2×20 znaków, któremu do poprawnej pracy wystarczy 40 bajtów tej pamięci), to nadmiarowe komórki pamięci można wykorzystać w sposób dowolny. Przyporządkowanie położenia znaków na wyświetlaczu adresom pamięci DD RAM zależy od tego, czy wyświetlacz jest jedno-, czy dwuwierszowy i czy jest włączona funkcja tzw. przesuwania okna wyświetlacza. W przypadku wyświetlacza dwuwierszowego, po wyzerowaniu wyświetlacza (przesuwanie okna wyłączone) w lewym górnym rogu jest wyświetlany znak o kodzie umieszczonym w DD RAM pod adresem 00h, kolejny znak odpowiada adresowi 01h itd. Dolny wiersz wyświetlacza (począwszy od lewej) ma kody znaków umieszczone w DD RAM pod adresami 40h, 41h itd. (rys. 10.11a).

Jeśli wyświetlacz jest jednowierszowy, kody pierwszej (lewej) połówki wyświetlacza są umieszczone pod adresami 00h, 01h itd., natomiast kody drugiej (prawej) połówki – pod adresami 40h, 41h itd., czyli przechodząc do drugiej połówki wyświetlacza jednowierszowego należy zmienić ustawienie adresu DD RAM tak samo, jak w przypadku przejścia do drugiej linii w wyświetlaczu dwuwierszowym (rys. 10.11b).

a)	00h	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh
	40h	41h	42h	43h	44h	45h	46h	47h	48h	49h	4Ah	4Bh	4Ch	4Dh	4Eh	4Fh

b)	00h	01h	02h	03h	04h	05h	06h	07h	40h	41h	42h	43h	44h	45h	46h	47h
----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Rys. 10.11. Początkowe (po wyzerowaniu wyświetlacza) adresy wyświetlanych znaków na przykładzie wyświetlacza (a) dwuwierszowego (2×16 znaków) oraz (b) jednowierszowego (1×16 znaków)

Poniżej przedstawiono skrócony opis poszczególnych komend sterujących wyświetlaczem (stan magistrali danych w kodzie binarnym; X – dowolny stan bitu):

Display clear ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 00000001)

Wypełnienie wyświetlacza spacjami, przejście w tryb zapisu danych wyświetlanych począwszy od pozycji w lewym górnym rogu wyświetlacza (dla wyświetlaczy dwuwierszowych; dla wyświetlaczy jednoliniowych jest to po prostu pozycja pierwszego znaku od lewej); wyłączenie trybu przesuwania okna wyświetlacza (jeśli tryb taki był włączony); maksymalny czas trwania operacji wynosi 1,64 ms.

Display/cursor home ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 0000001X)

Przejście w tryb zapisu danych wyświetlanych począwszy od pozycji w lewym górnym rogu wyświetlacza (dla wyświetlaczy dwuwierszowych; dla wyświetlaczy jednoliniowych jest to po prostu pozycja pierwszego znaku od lewej); wyłączenie trybu przesuwania okna wyświetlacza (jeśli tryb taki był włączony); maksymalny czas trwania operacji wynosi 1,64 ms.

Entry mode set ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 0000011S)

Określenie trybu pracy kursora/okna wyświetlacza:

- dla $S = 1$ przy kolejnych operacjach zapisu danych (znaków do wyświetlenia) kursor (miejsce wpisywania kolejnego znaku) nie zmienia położenia, natomiast przesuwa się cała zawartość wyświetlacza (tzw. przesuwanie okna wyświetlacza – efekt podobny do tego w kalkulatorach, z tym że jeśli wyświetlacz jest dwuliniowy, to przesuwana jest zawartość obydwu wierszy);
- dla $S = 0$ po każdej operacji zapisu danych (znaków do wyświetlenia) położenie kursora ulega zmianie, a przesuwanie okna wyświetlacza jest wyłączone;
- dla $I = 1$ kursor (okno wyświetlacza) przesuwa się w prawo (inkrementacja adresu zapisu kolejnego znaku);
- dla $I = 0$ kursor (okno wyświetlacza) przesuwa się w lewo (dekrementacja adresu zapisu kolejnego znaku).

Maksymalny czas trwania operacji wynosi 40 μ s.

Display ON/OFF ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 00001DCB)

Włączenie (dla $D = 1$) lub wyłączenie (dla $D = 0$) widoczności znaków na wyświetlaczu, kursora (widoczny dla $C = 1$, niewidoczny dla $C = 0$) i migania znaku w miejscu położenia kursora (włączone dla $B = 1$, wyłączone dla $B = 0$); maksymalny czas trwania operacji wynosi 40 μ s.

Display/cursor shift ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 0001SRXX)

Określenie trybu pracy kursora/okna wyświetlacza:

- dla $S = 1$ przy kolejnych operacjach zapisu danych (znaków do wyświetlenia) kursor nie zmienia położenia, natomiast przesuwa się cała zawartość wyświetlacza;
- dla $S = 0$ po każdej operacji zapisu danych (znaków do wyświetlenia) zmianie ulega położenie kursora, a przesuwanie okna wyświetlacza jest wyłączone;
- dla $R = 1$ kursor (okno wyświetlacza) przesuwa się w prawo;
- dla $R = 0$ kursor (okno wyświetlacza) przesuwa się w lewo.

Maksymalny czas trwania operacji wynosi 40 μ s.

Function set ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 001DN0XX)

Określenie trybu pracy magistrali danych i rodzaju wyświetlacza:

- dla $D = 1$ magistrala danych 8-bitowa,
- dla $D = 0$ magistrala danych 4-bitowa;
- dla $N = 1$ wyświetlacz dwuwierszowy;
- dla $N = 0$ wyświetlacz jednowierszowy.

Maksymalny czas trwania operacji wynosi 40 μ s.

CG RAM set ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 01AAALLL)

Określenie adresu pamięci generatora znaków, pod który nastąpi zapis (odczyt) danych operacją *Data write* lub *Data read*. AAA jest 3-bitowym adresem znaku definiowanego przez użytkownika. LLL jest 3-bitowym numerem poziomej linii (000 – położenie na samej górze, 111 – położenie na samym dole) składającej się na graficzne odwzorowanie znaku definiowanego przez użytkownika. Maksymalny czas trwania operacji wynosi 40 μ s.

DD RAM set ($RS = 0$, $R/\overline{W} = 0$, magistrala danych = 1AAAAAAA)

Określenie adresu pamięci wyświetlacza, pod który nastąpi zapis (odczyt) znaku operacją *Data write* lub *Data read*. AAAAAAA jest 7-bitowym adresem komórki pamięci wyświetlacza, w której przechowywane są kody quasi-ASCII wyświetlanych znaków. Czas trwania operacji wynosi 40 μ s.

Busy flag/address counter read ($RS = 0$, $R/\overline{W} = 1$, magistrala danych = BAAAAAAA)

Odczyt wskaźnika zajętości i wewnętrznego licznika adresu pamięci wyświetlacza. B jest stanem wskaźnika zajętości ($B = 1$ – wyświetlacz wykonuje poprzednią operację, $B = 0$ – wyświetlacz gotowy do wykonania następnej operacji), zaś AAAAAA jest 7-bitowym adresem pamięci wyświetlacza. Czas trwania operacji wynosi 0 μs .

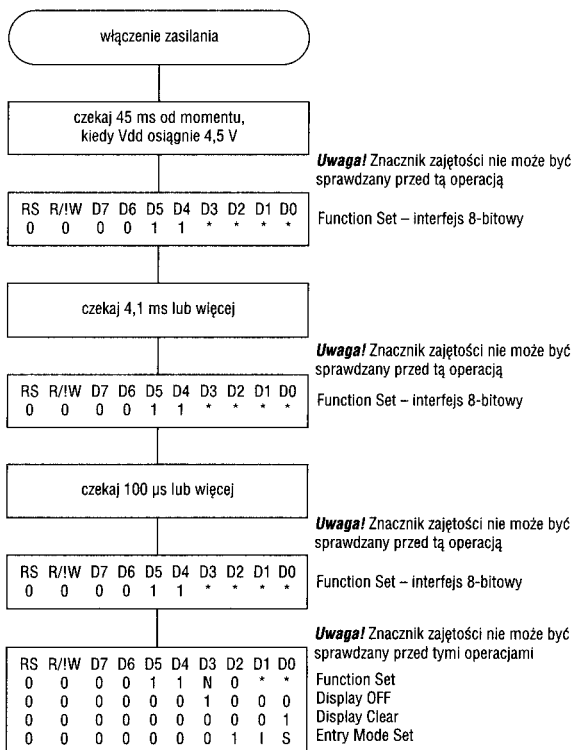
Data read ($RS = 1$, $R/\overline{W} = 1$, magistrala danych = odczytywany bajt danych)

Odczyt danych z pamięci znaków lub z pamięci kodów użytkownika w generatorze znaków (jeśli wydano poprzednio komendę *CG RAM set*). W przypadku odczytu pamięci generatora znaków istotnych jest tylko pięć młodszych bitów. Maksymalny czas trwania operacji wynosi 40 μs .

Data write ($RS = 1$, $R/\overline{W} = 0$, magistrala danych = zapisywany bajt danych)

Zapis danych do pamięci znaków lub do pamięci kodów użytkownika w generatorze znaków (jeśli wydano poprzednio komendę *CG RAM set*). W przypadku zapisu pamięci generatora znaków istotnych jest tylko pięć młodszych bitów. Maksymalny czas trwania operacji wynosi 40 μs .

Prawidłowa praca alfanumerycznego wyświetlacza LCD wymaga przeprowadzenia odpowiedniej procedury zerowania wyświetlacza (ich przebieg pokazano na rys. 10.12 i 10.13). Wprowadzie układ scalony sterujący wyświetlaczem ma wbudowany mechanizm autozerowania, jednak do poprawnego zadziałania wymaga on m.in.

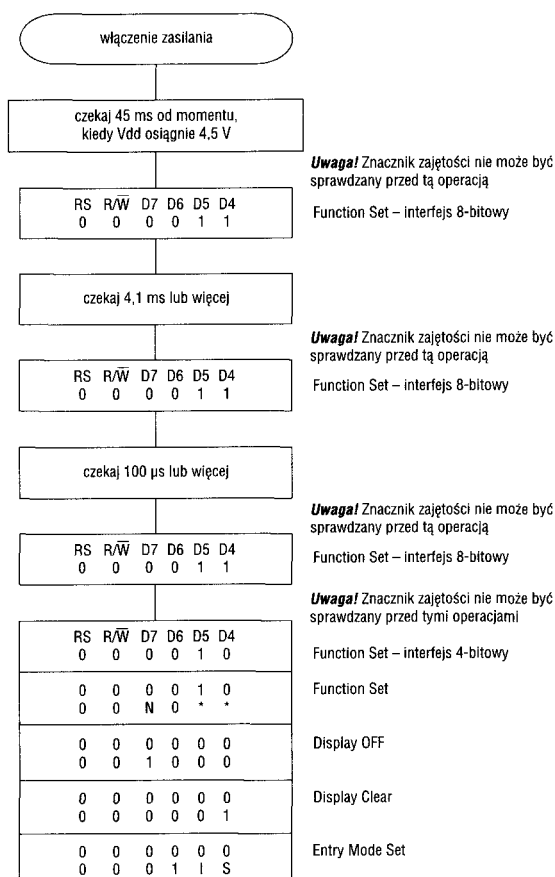


Rys. 10.12. Algorytm programowego zerowania alfanumerycznego wyświetlacza LCD pracującego w trybie komunikacji 4-bitowej

odpowiedniej szybkości narastania napięcia zasilającego. Ponadto autozerowanie identyfikuje wyświetlacz jako jednowierszowy (bez względu na stan faktyczny) i wprowadza go w tryb pracy z 8-bitową magistralą danych. Biorąc pod uwagę, że takie ustawienia standardowe nie zawsze odpowiadają rzeczywistości oraz to, że autozerowanie może nie zadziałać poprawnie, lepiej jest zawsze przeprowadzić programowe zerowanie wyświetlacza.

Programowe zerowanie wyświetlacza wymaga wykonania odpowiedniej, przedstawionej dalej, sekwencji działań:

- odczekać co najmniej 45 ms od momentu, gdy napięcie zasilania osiągnie wartość 4,5 V,
- wysłać komendę *Function Set* z parametrem $D = 1$ ($RS = 0$, $R/\overline{W} = 0$, magistrali danych = 0011XXXX),
- odczekać co najmniej 4,1 ms,
- wysłać komendę *Function Set* z parametrem $D = 1$ ($RS = 0$, $R/\overline{W} = 0$, magistrali danych = 0011XXXX),



Rys. 10.13. Algorytm programowego zerowania alfanumerycznego wyświetlacza LCD pracującego w trybie komunikacji 4-bitowej



Niedozwolone jest wykonywanie operacji sprawdzania znacznika zajętości wyświetlacza *Busy flag/address counter read* przed zakończeniem wykonywania operacji z punktu (f). Komendy z punktów (b), (d) oraz (f) należy wysłać z pojedynczym strobem linii sterującej E, nawet jeśli magistrala danych jest 4-bitowa; pozostałe komendy standardowo, w zależności od szerokości magistrali danych (czyli z podziałem na połówki bajtów i podwójnym strobem E jeśli magistrala danych jest 4-bitowa).

- e) odczekać co najmniej 100 μ s,
 - f) wysłać komendę *Function set* z parametrem D = 1 (RS = 0, R/\overline{W} = 0, magistrali danych = 0011XXXX),
 - g) wysłać komendę *Function set* z odpowiednimi parametrami D i N,
 - h) wysłać komendę *Display off* (RS = 0, R/\overline{W} = 0, magistrala danych = 00001000),
 - i) wysłać komendę *Display clear* (RS = 0, R/\overline{W} = 0, magistrala danych = 00000001),
 - j) wysłać komendę *Entry mode set* z odpowiednimi parametrami I oraz S.
- Przykładowe procedury obsługi alfanumerycznego wyświetlacza LCD pokazano na list. 10.3.

List. 10.3. Przykładowe procedury obsługi alfanumerycznego wyświetlacza LCD ze sterownikiem HD44780

```

;=====
; podstawowe procedury obsługi 2-wierszowego
; alfanumerycznego wyświetlacza LCD
; komunikacja 8-bitowa (dowolny port mikrokontrolera)
; udostępniane są procedury jak w deklaracji PUBLIC
; wymagane zdefiniowanie sposobu podłączenia LCD
; (typowo w module głównym) - parametry jak w EXTERN
;=====

PUBLIC init_LCD, zapisz_znak_LCD, LCD_XY, zapisz_string_LCD
EXTERN E_LCD, RW_LCD, RS_LCD, port_danych_LCD

;=====
; procedura pomocnicza, realizuje duże opóźnienie
; wykorzystywana tylko podczas inicjalizacji LCD
; przy wywołaniu w DPTR musi być wartość opóźnienia
; niszczy DPTR
;=====

delay:                                ; dla 12 MHz zegara
    DJNZ DPL, delay                  ; przybliżony czas opóźnienia
    DJNZ DPH, delay                  ; to 2*DPTR [us]
    RET

;=====
; procedura pomocnicza, realizuje krótkie opóźnienie
; przy wywołaniu w ACC musi być wartość opóźnienia
; niszczy ACC
;=====

delay2:
    DJNZ ACC, delay2                ; dla 12 MHz zegara
    RET                              ; przybliżony czas opóźnienia to 2*ACC [us]

```

```

;=====
; procedura zapisu pojedynczego znaku na LCD
; pod aktualne położenie kursora
; znak do zapisu przy wywołaniu musi być w ACC
; niszczy ACC
;=====

zapisz_znak_LCD:                ; znak do zapisu w ACC
    PUSH ACC                    ; znak do wysłania na stos
    CALL check_busy             ; czeka na możliwość dostępu do
                                ; wyświetlacza
    POP port_danych_LCD         ; znak do wysłania ze stosu od razu na port
                                ; LCD

zapis_danych:
    CLR RW_LCD                  ; odpowiednie ustawienie linii sterujących
    SETB RS_LCD
    SETB E_LCD                  ; i machnięcie strobem E
    CLR E_LCD
    RET

;=====
; procedura zapisu na wyświetlaczu LCD stringu
; zdefiniowanego w pamięci programu mikrokontrolera
; pod aktualne położenie kursora LCD
; znakiem końca stringu jest #
; przy wywołaniu w DPTR musi być wskaźnik (adres) stringu
; niszczy ACC, DPTR
;=====

zapisz_string_LCD:
    CLR A                        ; żeby następna instrukcja miała zerowy
                                ; offset
    MOVC A, @A+DPTR              ; pobranie znaku do wyświetlenia z pamięci
                                ; programu
    CJNE A, #'#', nastepny_znak  ; jeśli znak końca stringu (#),
    RET                          ; to powrót z procedury
nastepny_znak:                  ; w przeciwnym razie zapis kolejnego znaku
    PUSH DPH
    PUSH DPL                    ; chowa na stos DPTR, bo jest niszczone
                                ; przez
    CALL zapisz_znak_LCD         ; tę właśnie procedurę zapisu znaku na LCD
    POP DPL
    POP DPH                    ; odzyskanie DPTR
    INC DPTR                    ; żeby wskaźnik DPTR pokazywał na kolejny
                                ; znak
    JMP zapisz_string_LCD

;=====
; procedura ustawienia kursora LCD w zadanym miejscu
; na ekranie
; przy wywołaniu w ACC zadane miejsce kursora
; zdefiniowane jak adres DD RAM wyświetlacza
; (czyli np. dla 2-wierszowego LCD pierwszy znak
; w dolnym wierszu ma adres 40H)
; niszczy ACC, DPTR
;=====

LCD_XY:
    SETB ACC.7                  ; modyfikacja bitu D7 odpowiednio dla
                                ; komendy Set DD RAM Address
    PUSH ACC                    ; przyszła pozycja kursora na stos
    CALL check_busy             ; czekanie na możliwość dostępu do

```

```

                                ; wyświetlacza
POP port_danych_LCD           ; schowany na stos argument od razu na port
CALL zapis_komendy             ; wysłanie komendy Set DD RAM Address
RET

;=====
; procedura sprawdzania zajętości wyświetlacza LCD
; czeka, aż do momentu zwolnienia dostępu do LCD
; niszczy ACC
;=====

check_busy:
    MOV port_danych_LCD, #0FFH      ; port do odczytu
    SETB RW_LCD                     ; ustawienie odczytu
    CLR RS_LCD                       ; ustawienie RS = 0
sprawdz_ponownie:
    SETB E_LCD                       ; strob E w stan aktywny
    MOV A, port_danych_LCD           ; odczyt szyny danych LCD
    CLR E_LCD                       ; strob E w stan nieaktywny
    JB ACC.7, sprawdz_ponownie      ; jeśli BF = 1, to czekaj (skok)
    RET

;=====
; procedura pomocnicza
;=====

zapis_komendy:
    CLR RW_LCD                       ; odpowiednie ustawienie linii sterujących
    CLR RS_LCD
    SETB E_LCD                       ; i machnięcie strobem
    CLR E_LCD
    RET

;=====
; procedura programowej inicjalizacji wyświetlacza LCD
; niszczy DPTR, ACC
;=====

init_LCD: .
    CLR E_LCD
    MOV DPTR, #0FFFFH               ; ma być opóźnienie ok. 20 ms,
                                      ; jest ok. 100 ms

    CALL delay
    MOV port_danych_LCD, #30H
    CALL zapis_komendy
    MOV DPTR, #0FFFFH               ; ma być opóźnienie ok. 4,2 ms,
                                      ; jest ok. 100 ms

    CALL delay
    CALL zapis_komendy
    MOV DPTR, #0FFFFH               ; ma być opóźnienie ok. 100 us,
                                      ; jest ok. 100 ms

    CALL delay
    CALL zapis_komendy
    MOV DPTR, #0FFFFH               ; ma być opóźnienie ok. 100 us,
                                      ; jest ok. 100 ms

    CALL delay
    MOV port_danych_LCD, #38H       ; 8-bitowy interfejs, LCD
                                      ; 2-wierszowy

    CALL zapis_komendy
    CALL check_busy
    MOV port_danych_LCD, #8H        ; komenda Display Off
    CALL zapis_komendy

```

```

CALL check_busy
MOV port_danych_LCD, #1H          ; komenda Display Clear
CALL zapis_komendy
CALL check_busy
MOV port_danych_LCD, #6H          ; autoinkrementacja położenia
                                   ; kursora

CALL zapis_komendy
CALL check_busy
MOV port_danych_LCD, #0FH         ; komenda Display On
                                   ; z kursorem widocznym
CALL zapis_komendy                ; i migoczącym

RET

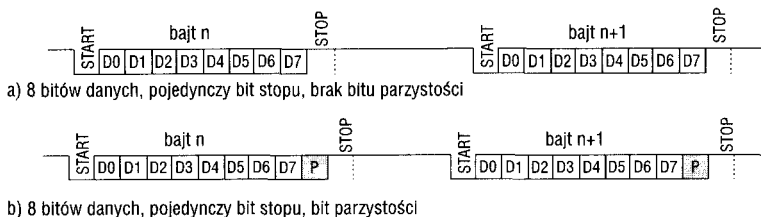
END                                ; koniec modułu

```

10.6. Interfejs RS232

Jednym z najczęściej wykorzystywanych interfejsów, zwłaszcza w systemach mikroprocesorowych budowanych na mikrokontrolerach, jest RS232. W standardzie RS232 dane są przesyłane asynchronicznie, czyli nie występuje tam jawny sygnał zegarowy – odbiornik musi wykryć początek transmisji na podstawie pierwszego zbocza nadchodzącej paczki danych. Nieaktywnym stanem interfejsu jest jedynka logiczna. Rozpoczęcie transmisji polega na wysłaniu tzw. bitu startu (zero logiczne), po którym następują kolejne bity danych (począwszy od najmniej znaczącego), opcjonalny bit parzystości i na końcu bit stopu (jedynka logiczna). Wygląd typowej ramki danych pokazano na **rys. 10.14**. Czas trwania wszystkich bitów jest identyczny i powinien być dostosowany do standardowych prędkości transmisji. Najczęściej stosowane standardowe prędkości transmisji w standardzie RS232 to 1200, 2400, 4800, 9600, 19200, 57600 i 115200 bodów (czyli bitów na sekundę). Jest to prędkość rzeczywista, uwzględniająca wszystkie przesyłane bity (czyli także startu, stopu i ewentualnie parzystości), podczas gdy efektywna prędkość transmisji danych jest o co najmniej 20 procent mniejsza (przy założeniu zerowych odstępów między kolejnymi paczkami danych i transmisji bez bitu parzystości). Czas trwania pojedynczego bitu przy danej prędkości transmisji jest obliczany na podstawie szybkości rzeczywistej, czyli dla 1200 bodów będzie wynosił $1\text{ s}/1200 = 0,833(3)\text{ ms}$.

Przy konstruowaniu urządzeń wykorzystujących RS232 należy pamiętać o tym, że poziomy napięcie linii sygnałowych w tym standardzie nie są kompatybilne z poziomami TTL/CMOS. Jedynce logicznej na linii danych odpowiada napięcie z zakresu -3 do -15 V , natomiast zeru logicznemu – napięcie z zakresu $+3$ do $+15\text{ V}$. Napięcia z zakresu od -3 do $+3\text{ V}$ są traktowane jako stan przejściowy i odpowiadają nieokreślonemu stanowi logicznemu. Wprawdzie niektóre układy konwersji poziomów traktują napięcia

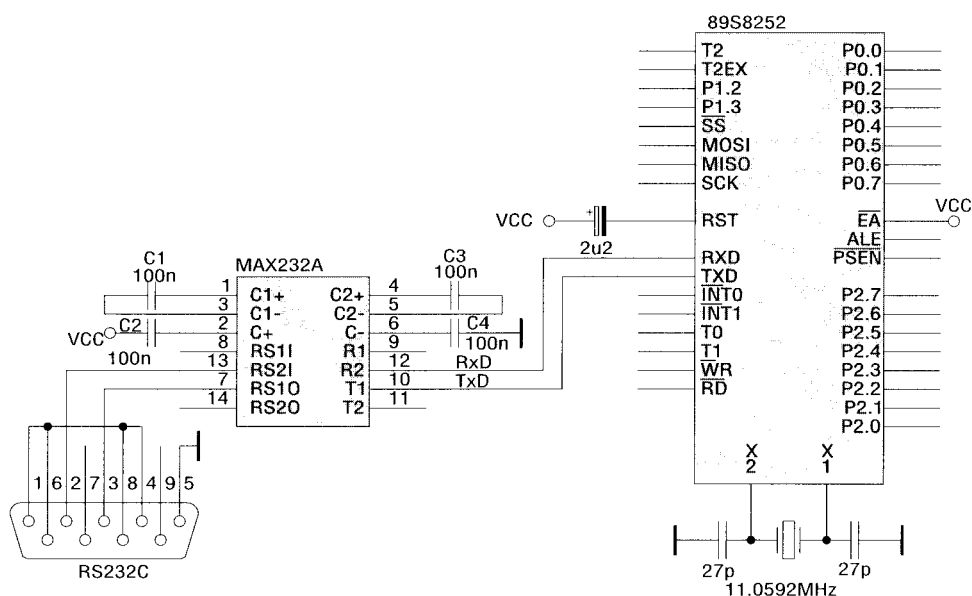


Rys. 10.14. Przykłady typowych transmisji asynchronicznych

w bliskiej okolicy potencjału masy jeszcze jako jedynek logiczną, nie jest to jednak reguła, na której można by się było oprzeć, zwłaszcza w przypadku konstrukcji komercyjnych, czy nawet amatorskich, ale przewidzianych do nieco szerszego stosowania. W celu dopasowania poziomów napięć układów logicznych TTL/CMOS (jakie w szczególności wykorzystują opisywane mikrokontrolery rodziny '51) do poziomów standardu RS232 wykorzystuje się najczęściej gotowe konwertery poziomów logicznych, z których najbardziej popularny jest układ MAX232 (oraz jego zmodyfikowana wersja - MAX232A - umożliwiająca pracę z większymi prędkościami transmisji i kondensatorami o mniejszych pojemnościach).

Implementację interfejsu RS232 w konstruowanym urządzeniu najłatwiej jest wykonać wykorzystując do tego celu łącze szeregowe, jakie można znaleźć w większości mikrokontrolerów. Należy wówczas dołączyć do odpowiednich wyprowadzeń mikrokontrolera układ konwertera poziomów (patrz np. rys. 10.15) i odpowiednie złącze (obecnie najczęściej wykorzystuje się do tego celu wtyk DB 9). Potem wystarczy już tylko ustawić prędkość transmisji (zazwyczaj przy pomocy układu licznikowego T1 lub T2 - patrz rozdz. 3.1) i napisać odpowiednie procedury odpowiedzialne za wysyłanie i odbiór danych.

Jeśli nie można skorzystać z wewnętrznego sprzętowego interfejsu szeregowego (bo go po prostu w mikrokontrolerze nie ma, albo jest już wykorzystywany do innych celów) warto rozważyć programową realizację takiego interfejsu. Implementacja nadajnika jest dość prosta i możliwa nawet dla stosunkowo dużych prędkości transmisji, ponieważ sprowadza się jedynie do odpowiedniej zmiany stanu wybranego wyprowadzenia dokonywanej zazwyczaj w takt przerwań od jednego z układów licznikowych (list. 10.4). Nieco gorzej ma się rzecz z programowym odbiornikiem łącza szeregowego, który musi przede wszystkim wykryć niesynchronizowane z pracą mikrokontrolera, czy jego wewnętrznych liczników, opadające zbocze bitu startu przychodzącej ramki danych. Możliwe są w tym



Rys. 10.15. Przykładowy sposób dołączenia do mikrokontrolera konwertera poziomów RS232 <-> TTL/CMOS

przypadku dwa podejścia do zagadnienia: detekcja zbocza w oparciu o wejście przerwań zewnętrznych lub programowe testowanie stanu wyprowadzenia z częstotliwością wyraźnie wyższą od stosowanej prędkości transmisji. Przykład programowej implementacji odbiornika łączy szeregowego zamieszczono w **list. 10.5**.

Dość istotnym problemem w przypadku realizacji interfejsu RS232 (czy jakiegokolwiek innego interfejsu asynchronicznego) jest tolerancja częstotliwości użytej do taktowania łączy szeregowych. Układ synchronizacji generatora taktującego odbiornik wymusza próbkowanie bitu startu mniej więcej w połowie tego bitu. Przyjmując, że próbkowanie następuje dokładnie w połowie bitu, odbiór przesyłanej informacji jest poprawny dopóki ostatecznie zbocze próbujące ostatni bit stopu będzie przesunięte (w stosunku do idealnej zgodności prędkości transmisji komunikujących się ze sobą urządzeń) nie więcej niż o połowę czasu trwania pojedynczego bitu.

Względna dopuszczalna tolerancja (niedokładność) zestrojenia generatorów taktujących łączy szeregowych komunikujących się ze sobą urządzeń wynosi zatem $50/N$ [%], gdzie N jest liczbą bitów transmitowanych w pojedynczej paczce (uwzględniając bit startu, stopu i parzystości). Dla przykładu przy transmisji 8 bitów danych bez bitu parzystości wymagana współbieżność generatorów nie może być gorsza niż 5,0%. W rzeczywistości musi być ona lepsza, ponieważ próbkowanie bitu startu nigdy nie zachodzi idealnie w jego środku (zwłaszcza w przypadku odbiorników programowych), a ponadto należy uwzględnić skończony czas narastania zbocza i zniekształcenia przesyłanego sygnału. Dodatkowym utrudnieniem, wymagającym znacznie lepszego zestrojenia generatorów, jest transmisja większych ilości danych z zerowymi przerwami między przysyłaniem kolejnych bajtów, gdyż w takim przypadku, jeśli wolniejszy generator występuje w odbiorniku, nie ma on czasu na odrobienie spóźnienia jakie powstało przy transmisji pierwszego bajtu przed rozpoczęciem odbioru następnego. Tym samym owo opóźnienie (w sensie utraty synchronizacji między urządzeniem nadającym i odbierającym) powiększa się coraz bardziej podczas odbioru kolejnych bajtów. Skuteczną techniką zapobiegającą utracie synchronizacji jest w takich przypadkach nadawanie ze zwiększoną (np. do dwóch) liczbą bitów stopu, przy odbiorze ustawionym na pojedynczy bit stopu. Nadawany dodatkowy bit stopu zwiększa odstęp między kolejnymi paczkami danych i umożliwia wolniejszemu urządzeniu odbierającemu całkowite zakończenie obsługi transmisji jednego bajtu przed pojawieniem się bitu startu rozpoczynającego transmisję kolejnego bajtu.

List. 10.4. Programowa implementacja nadajnika RS232

```
; =====
; procedura programowej obsługi nadajnika łączy RS
; bez parzystości, z podwójnym bitem stopu
; musi być wywoływana z częstotliwością równą baud rate (prędkości
; transmisji)
;     TX_aktywny - bajt pomocniczy, zlicza liczbę bitów do wysłania
;     TX_buf - jednobajtowy bufor, w którym umieszczono bajt do
; wysłania
; procedura umieszczająca bajt do wysłania musi również ustawić bit
; „zaczni_j_nadawanie”, który jest automatycznie zerowany pod koniec
; nadawania
; udostępniane są procedury jak w deklaracji PUBLIC
; wymagane zdefiniowanie (typowo w module głównym) parametrów jak
; w EXTERN
;     niszczy ACC i CY
; =====
```

```

PUBLIC nadawanie_RS
EXTERN zacznij_nadawanie, TX_aktywny, TX_pin, TX_buf

nadawanie_RS:
    MOV A, TX_aktywny
    JNZ nastepny_bit      ; skok, jeśli w trakcie wysyłania bajtu
    JB zacznij_nadawanie, bit_startu
    SETB TX_pin
    RET

bit_startu:
    MOV TX_aktywny, #10   ; 11 bitów do wysłania (oprócz startu)
    CLR TX_pin            ; bit startu
    RET

nastepny_bit:
    DEC TX_aktywny        ; dekrementacja TX_aktywny
    CJNE A, #2, moze_drugi_bit_stopu
    SETB TX_pin           ; wysłanie pierwszego bit stopu
    RET

moze_drugi_bit_stopu:
    CJNE A, #1, bit_danych
    SETB TX_pin           ; wysłanie drugiego bitu stopu
    CLR zacznij_nadawanie ; i zerowanie znacznika nadawania
    RET

bit_danych:
    MOV A, TX_buf         ; wysłanie kolejnego bitu danych
    MOV C, ACC.0
    MOV TX_pin, C
    RR A
    MOV TX_buf, A
    RET

END                                ; koniec modułu

```

List. 10.5. Programowa implementacja odbiornika RS232

```

; *****
; procedura programowego odbiornika łączy RS
; musi być wywoływana z częstotliwością 1_szpilek x baud rate
; bit sygnalizujący Frame Error jest zwracany stanem wskaźnika
; F0 (PSW.5)
; udostępniane są procedury jak w deklaracji PUBLIC
; wymagane zdefiniowanie (typowo w module głównym) parametrów jak
; w EXTERN
;   niszczy ACC, PSW (CY, F0)
; *****

EXTERN RX_gotowy, l_szpilek, polowa_l_szpilek
EXTERN RX_aktywny, RX_pin, RX_buf, RX_temp, nr_szpilki
PUBLIC odbior_RS

odbior_RS:
    MOV A, RX_aktywny
    JNZ obsluga_RX                ; jeśli RX_aktywny <> 0
                                    ; to obsługa
                                    ; jak nie, to może bit startu?
    JB RX_pin, koniec            ; powrót, jeśli brak bitu startu
    MOV RX_aktywny, #10         ; do odbioru jest 10 bitów

```

```

MOV nr_szpilki, #1_szpilek
RET

obsługa_RX:
    DJNZ nr_szpilki, dalej_RX          ; -1 i skok, jeśli
                                        ; nr_szpilki <> 0
    MOV nr_szpilki, #1_szpilek
koniec:
    CLR F0                            ; zerowanie wskaźnika Frame
                                        ; Error
    RET

dalej_RX:
    MOV A, nr_szpilki
    CJNE A, #polowa_1_szpilek, koniec ; powrót, jeśli nie połowa bitu
    DJNZ RX_aktywny, wczytaj_bit_danych ; -1 i skok, jeśli
                                        ; RX_aktywny <> 0
    JNB RX_pin, blad_ramki             ; test bitu stopu, jeśli OK to
    MOV RX_buf, RX_temp                ; RX_temp do bufora odbiornika
    CLR F0                             ; sygnalizacja, że stop OK
    SETB RX_gotowy                     ; i że dane czekają w buforze
    RET

blad_ramki:
    SETB F0                            ; sygnalizacja Frame Error
                                        ; (błędu ramki)
    RET

wczytaj_bit_danych:
    MOV A, RX_aktywny
    CJNE A, #9, nie_bit_startu         ; skok, jeśli RX_aktywny <> 9
    JNB RX_pin, koniec                 ; poprawny bit startu
    MOV RX_aktywny, #0                 ; zakłócenie, a nie bit startu
    RET

nie_bit_startu:
                                        ; doklejenie kolejnego
                                        ; odebranego bitu
    MOV A, RX_temp
    MOV C, RX_pin
    RRC A
    MOV RX_temp, A
    CLR F0                             ; i zerowanie wskaźnika błędu
                                        ; ramki
    RET

END                                    ; koniec modułu

```

10.7. Sterowanie układami wyposażonymi w interfejs SPI

Interfejs SPI (patrz rozdz. 3.3.5) jest bardzo popularnym interfejsem synchronicznym stosowanym w wielu układach peryferyjnych (np. w przetwornikach A/C i C/A). Typowe przebiegi występujące na magistrali SPI podczas wpisywania danych do przetwornika C/A MAX552 pokazano na rys. 10.16.

W przypadku układów rodziny '51 niewiele typów mikrokontrolerów ma wbudowany sprzętowy interfejs SPI, nie powinno to jednak stanowić większej przeszkody, gdyż jego realizacja programowa jest stosunkowo prosta. Przykład procedur pokazujących obsługę interfejsu SPI w układzie obsługującym przetworniki jak na rys. 10.17 i przykładową technikę podejścia implementacji programowej tego interfejsu przedstawiono w list. 10.6.


```

sclk equ p1.7
miso equ p1.6
mosi equ p1.5

; =====
; procedura sterująca przetwornikiem A/C (wyzwolenie pomiaru
; na zadanym wejściu przetwornika i odczyt wyniku)
; na wejściu do procedury w R7 numer wejścia analogowego,
; na którym należy dokonać pomiaru napięcia
; przy wyjściu procedury starszy bajt wyniku w R6, młodszy w R7
; niszczy ACC, PSW
; =====

_get_adc:
    CLR adc_cs          ; zezwolenie na komunikację
                        ; z przetwornikiem A/C
    MOV SPCR, #01010001B ; ustawienie trybu pracy interfejsu SPI
    MOV A, #6           ; tryb pracy przetwornika: single-ended
    CALL SPI_transfer    ; wysłanie pierwszego bajtu (sterującego)
    MOV A, R7            ; numer wejścia analogowego
    RR A                 ; na dwóch najbardziej znaczących bitach
    CALL SPI_transfer    ; wysłanie drugiego bajtu (numera wejścia
                        ; analogowego)
    ANL A, #0FH          ; maskowanie nieznaczących bitów wyniku
    MOV R6, A            ; bardziej znaczący bajt wyniku do R6
    CALL SPI_transfer    ; odczyt mniej znaczącego bajtu
    MOV R7, A            ; mniej znaczący bajt wyniku do R6
    SETB adc_cs          ; zablokowanie dalszej komunikacji
    RET

; =====
; procedura sterująca przetwornikiem C/A
; na wejściu do procedury w R7 numer wyjścia analogowego,
; w R5 wartość binarna napięcia wyjściowego
; niszczy ACC, PSW
; =====

_set_dac:
    CLR dac_cs          ; zezwolenie na komunikację
                        ; z przetwornikiem
    MOV SPCR, #01010001B ; ustawienie trybu pracy interfejsu SPI
    MOV A, R7            ; wejściowe
    RL A                 ; wstępna obróbka dopasowująca dane
                        ; do formatu wymaganego przez przetwornik
    ADD A, R7            ; słowo sterujące przetwornikiem C/A
    XRL A, #00100001B    ; (numer wyjścia)
                        ; i jego wysłanie
    CALL SPI_transfer
    MOV A, R5            ; wysłanie właściwych danych (napięcia)
    CALL SPI_transfer
    SETB dac_cs
    RET

; =====
; procedura odpowiedzialna za obsługę sprzętowej
; wersji interfejsu SPI
; =====

SPI_transfer:
    MOV SPDR, A          ; wysyłana jest zawartość ACC
    MOV A, SPSR          ; sprawdzanie w pętli bitu stanu SPI
    JNB ACC.7, $-2       ; w oczekiwaniu na koniec transmisji
    MOV A, SPDR          ; odebrany bajt będzie w ACC
    RET

```

```

; =====
; i analogiczna procedura dla implementacji
; programowej, z tym że przystosowana wyłącznie
; do wysyłania danych (czyli do sterowania
; przetwornikiem C/A)
; =====
; np.
    DOUT EQU P1.4
    SCLK EQU P1.5

out_spi:
    MOV R7, #8                ; do wysłania jest 8 bitów
loop1:
    MOV C, ACC.7              ; kolejny bit do CY
    MOV DOUT, C               ; i na wyprowadzenie DOUT mikrokontrolera
                                ; służące jako linia MOSI

    SETB SCLK
    CLR SCLK                  ; machnięcie impulsem zegara na linii SCLK
    RL A                      ; następny bit do wysłania na ACC.7
    DJNZ R7, loop1            ; jeśli nie wszystkie bity, to skok
    RET                      ; a jeśli wszystkie, powrót z procedury

END                            ; koniec modułu

```

10.8. Interfejs I²C

W systemach mikroprocesorowych budowanych z wykorzystaniem mikrokontrolerów jednoukładowych bardzo często zachodzi potrzeba dołączenia układów peryferyjnych, które nie wymagają częstej komunikacji z CPU. Przykładami takich układów mogą być:

- pamięć EEPROM zawierająca nastawy początkowe urządzenia, współczynniki kalibracyjne przyrządu pomiarowego itp.,
- zegar czasu rzeczywistego,
- sterownik wyświetlacza LED lub LCD,
- kontroler klawiatury,
- specjalizowane układy analogowe o programowo zmienianych parametrach pracy, np. syntezer częstotliwości, cyfrowo przestrajany filtr, wzmacniacz itp.

Rzadka wymiana informacji między mikrokontrolerem, a takimi układami pozwala na jej realizację za pomocą interfejsu szeregowego. Przesyłanie informacji z zastosowaniem interfejsu szeregowego następuje przy użyciu niewielkiej liczby linii sygnałowych. Stąd też stosowanie interfejsu szeregowego ma, w porównaniu do interfejsu równoległego, wiele zalet:

- a) Jeśli wszystkie układy peryferyjne wykorzystujące interfejs równoległy zostaną zastąpione układami z interfejsem szeregowym, to wyprowadzenia mikrokontrolera służące do komunikacji z obszarem zewnętrznej pamięci danych (linie portów P0 i P2, wyprowadzenia sygnałów \overline{WR} i \overline{RD}) mogą być wykorzystane do innych celów.
- b) Zmniejszenie liczby linii sygnałowych prowadzi do zredukowania liczby wyprowadzeń układów peryferyjnych, a w konsekwencji do zmniejszenia kosztu tych elementów (tańsze obudowy) i całego układu (mniejsza płytka).
- c) Upraszcza się istotnie projekt płytki drukowanej (prowadzone są tylko 2...3 linie sygnałowe zamiast kilkunastu), zwłaszcza w przypadku wielu układów wykorzystujących tę samą szynę interfejsu.

- d) Możliwe jest łatwe i tanie sprzęganie układów umieszczonych na różnych płytach (typowym przykładem jest tu oddzielna płytka wyświetlacza). Niski koszt wynika głównie ze stosowania małych złącz (pojedyncze linie sygnałowe) oraz braku buforów (przy małych prędkościach transmisji pojemności montażowe mają znikomy wpływ na pracę interfejsu).
- e) Właściwość wymieniona w punkcie poprzednim (d) powoduje, że przy minimalnym nakładzie środków dodatkowych szyna może być wykorzystana do celów diagnostycznych, czyli do testowania i uruchamiania urządzenia.

Nie każdy interfejs szeregowy będzie jednak dobrze spełniał rolę wspólnej szyny służącej do sterowania wewnętrznymi (w sensie urządzenia, a nie mikrokontrolera) układów peryferyjnych. Standardowe łącze szeregowe występujące w mikrokontrolerach rodziny 51 doskonale nadaje się np. do realizacji komunikacji wieloprocesorowej, ale nie ma ono mechanizmów, które pozwoliłyby na łatwe przystosowanie go do roli interfejsu w układach peryferyjnych nie mających wbudowanej inteligencji (w postaci mikrokontrolera). Rodzajem interfejsu szeregowego dobrze funkcjonującego w takich zastosowaniach jest natomiast szyna I²C.

10.8.1. Ogólne założenia interfejsu I²C

Szyna I²C charakteryzuje się następującymi cechami:

- transmisja odbywa się za pomocą dwóch linii sygnałowych: SDA - linii danych i SCL - linii sygnału taktującego,
- możliwe jest przesyłanie informacji między wieloma urządzeniami wykorzystującymi wspólną szynę,
- transmisja danych jest dwukierunkowa (nadawanie i odbiór),
- każdy z układów dołączonych do szyny ma niepowtarzalny (jedyny w całym systemie) adres, umożliwiający jednoznaczne zaadresowanie danego urządzenia do nadawania lub odbioru,
- przesyłaniem danych steruje układ nadrzędny, przy czym w systemie może występować więcej niż jeden układ nadrzędny,
- w przypadku jednoczesnego rozpoczęcia nadawania przez kilka urządzeń nadrzędnych, mechanizm arbitrażu nie dopuszcza do utraty, ani przekłamania przesyłanych danych,
- synchronizacja transmisji za pomocą sygnału taktującego umożliwia komunikację między urządzeniami stosującymi różne prędkości przesyłania danych,
- maksymalna prędkość transmisji danych po szynie I²C wynosi 100 kbd w standardowym trybie pracy szyny i 400 kbd w trybie szybkim,
- wszystkie układy posiadające interfejs I²C są wyposażone w układy filtracji zakłóceń (w postaci szpilek napięciowych mogących występować na liniach SDA i SCL), istotnie podwyższające niezawodność pracy interfejsu,
- liczba układów podłączonych do szyny ograniczona jest wyłącznie maksymalną pojemnością szyny, równą 400 pF.

Łatwa i tania implementacja interfejsu I²C w układach peryferyjnych jest możliwa dzięki jednej z podstawowych cech szyny I²C, a mianowicie podziału urządzeń dołączonych do szyny na nadrzędne i podrzędne. Urządzenie nadrzędne jest układem

inicjalizującym transmisję po szynie, generującym sygnał taktujący, adresującym urządzenia podrzędne, określającym rodzaj transmisji (nadawanie/odbiór danych) itp. Urządzenie podrzędne musi zaś jedynie stwierdzić, czy zostało ono zaadresowane (porównać adres przesłany po szynie z własnym adresem urządzenia) oraz, w takt sygnału generowanego przez urządzenie nadrzędne, wysłać lub odebrać dane. Dzięki takiemu rozwiązaniu inteligencję musi mieć jedynie urządzenie nadrzędne (rolę tę pełni zwykle mikrokontroler), podczas gdy interfejs urządzenia podrzędnego może być zrealizowany jako prosty automat o niewielkiej liczbie stanów.

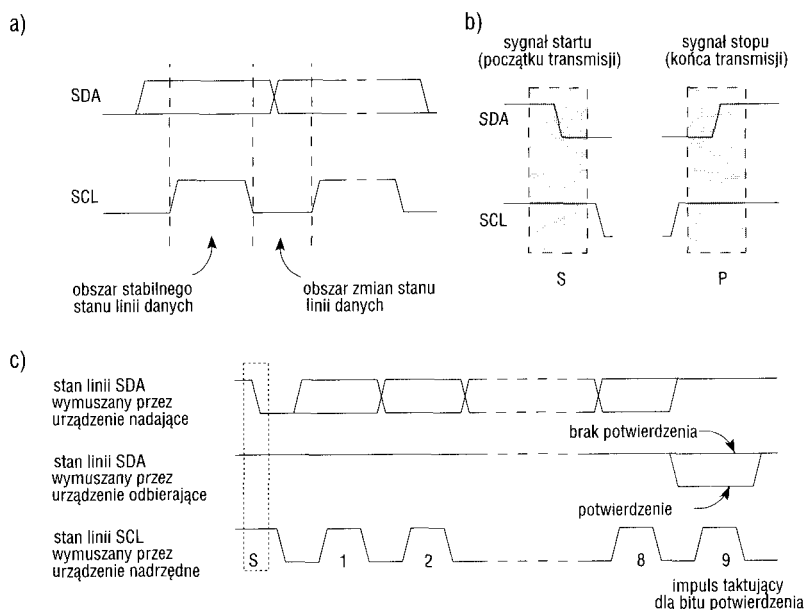
Z punktu widzenia konstruktora, interfejs I^2C ma wiele zalet wynikających bezpośrednio ze sposobu funkcjonowania szyny. Najistotniejszą z nich jest chyba to, że te same układy wyposażone w interfejs I^2C mogą być wykorzystywane w wielu różnych urządzeniach, a ich sterowanie zawsze będzie identyczne. Dzięki temu raz utworzone moduły biblioteczne mogą być wykorzystywane wielokrotnie, skracając poważnie czas opracowywania nowych konstrukcji. Ułatwione jest też opracowywanie wariantów tego samego urządzenia, ponieważ w wielu przypadkach będzie się ono sprowadzało do usunięcia, zamiany lub dołączenia nowego układu do istniejącej już szyny.

10.8.2. Przesyłanie informacji po szynie I^2C

Liniami sygnałowymi interfejsu I^2C są linie SDA i SCL. Są to linie dwukierunkowe podłączone do dodatniego napięcia zasilania za pośrednictwem rezystorów podciągających. Jeśli szyna jest zwolniona (brak transmisji), obie linie znajdują się w stanie wysokim. Wszystkie układy podłączone do szyny muszą mieć stopnie końcowe (linii SDA i SCL) typu otwarty kolektor lub otwarty dren – umożliwia to realizację funkcji AND typu otwarty kolektor (otwarty dren), wykorzystywanej m.in. przez mechanizm arbitrażu szyny. Ze względu na możliwość dołączania do szyny układów wykonanych w różnych technologiach (CMOS, NMOS, układy bipolarne) przedziały napięć odpowiadające poziomom logicznego zera (stan niski) i jedynki (stan wysoki) nie muszą być sztywno ustalone i mogą zależeć od wartości napięcia V_{DD} , do którego dołączone są rezystory podciągające linii SDA i SCL.

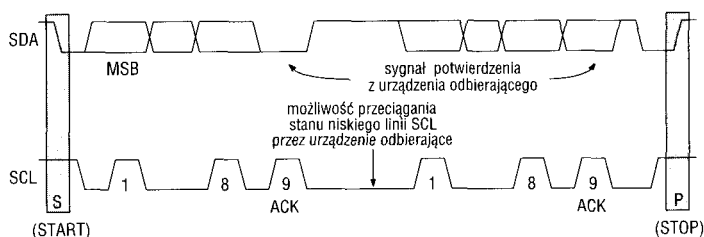
Podczas transmisji danych, impulsy taktujące na szynie SCL są wysyłane w ilości jeden impuls na każdy bit przesyłanych danych. Podczas wysokiego stanu na linii SCL stan linii SDA musi być stabilny – stan tej linii może ulegać zmianie tylko podczas niskiego stanu linii SCL (rys. 10.18a). Wyjątkiem od tej reguły są sygnały START i STOP wysyłane przez urządzenie nadrzędne (rys. 10.18b). Wysłanie sygnału START polega na wymuszeniu opadającego zbocza na linii SDA podczas trwania wysokiego stanu na linii SCL. Sygnał START służy do identyfikacji początku transmisji – po pojawieniu się tego sygnału szyna jest uważana za zajęta aż do momentu wystąpienia sygnału STOP, identyfikującego koniec transmisji. W rzeczywistości szyna jest uważana za zwolnioną dopiero po pewnym czasie od wykrycia sygnału STOP, czyli po stwierdzeniu pojawienia się narastającego zbocza na linii SDA przy wysokim stanie linii SCL.

Każdy bajt danych przesyłanych po szynie I^2C musi składać się z 8 bitów, przy czym dane są nadawane począwszy od bitu najbardziej znaczącego. Liczba bajtów danych przesyłanych w ramach danej transmisji nie jest ograniczona. Każda operacja przesłania bajtu danych powinna zakończyć się bitem potwierdzenia (rys. 10.18c). Impuls taktujący związany z bitem potwierdzenia jest generowany przez urządzenie nadrzędne.



Rys. 10.18. Sposób przesyłania informacji magistralą I²C: a) transmisja poszczególnych bitów; b) nadawanie sygnałów START i STOP; c) sygnalizacja potwierdzenia

Urządzenie nadające dane musi na czas bitu potwierdzenia zwolnić linię SDA (umożliwić jej przyjęcie stanu wysokiego), zaś urządzenie odbierające na czas trwania wysokiego stanu na linii SCL (oczywiście z uwzględnieniem stanów przejściowych) powinno wymusić stan niski na linii SDA. Jeśli zaadresowane urządzenie podrzędne nie jest w stanie potwierdzić swojego adresu, musi pozostawić ono linię SDA w stanie wysokim. Wówczas urządzenie nadrzędne może wysłać sygnał STOP i tym samym przerwać transmisję. Analogiczna sytuacja ma miejsce, gdy urządzenie podrzędne potwierdzi swój adres, ale nie jest w stanie odebrać jednego z następnych bajtów danych i chce wymusić przerwanie transmisji ze strony urządzenia nadrzędne. Urządzenie podrzędne powinno wówczas wysłać sygnał braku potwierdzenia (czyli po prostu pozostawić linię SDA w stanie wysokim na czas trwania bitu potwierdzenia). W odpowiedzi urządzenie nadrzędne wyśle sygnał STOP i zakończy transmisję danych. Jeśli dane są odbierane przez urządzenie nadrzędne, to wysłanie przez to urządzenie bitu braku potwierdzenia jest informacją dla urządzenia podrzędnego (nadającego dane),



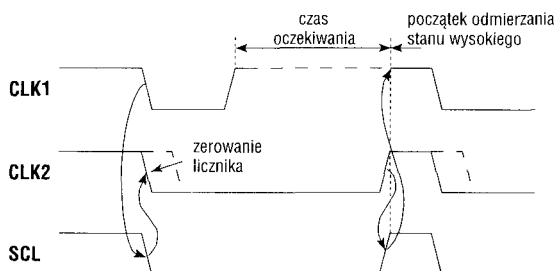
Rys. 10.19. Typowa transmisja danych po szynie I²C

że zakończono właśnie transmisję ostatniego bajtu danych. Urządzenie podrzędne musi w takim przypadku zwolnić linię danych (SDA), by umożliwić urządzeniu nadrzędnemu wysłanie sygnału STOP. Jeśli urządzenie odbierające dane nie może z pewnych względów rozpocząć odbioru następnego bajtu, to może ono przeciągnąć niski stan na linii SCL i wymusić w ten sposób na urządzeniu nadającym przejście w stan oczekiwania (rys. 10.19). Transmisja danych będzie w takim przypadku kontynuowana po zwolnieniu linii SCL przez urządzenie odbierające dane.

10.8.3. Mechanizm arbitrażu i synchronizacji

Możliwość podłączenia do szyny I²C kilku urządzeń nadrzędnych może być źródłem sytuacji, w której dwa lub więcej urządzeń nadrzędnych usiłuje rozpocząć transmisję danych jednocześnie. Skuteczne rozstrzyganie takich konfliktów jest możliwe dzięki wprowadzonemu w standardzie I²C mechanizmowi arbitrażu i synchronizacji. Mechanizm ten wykorzystuje fakt, iż stan linii sygnałowych interfejsu jest funkcją AND typu otwarty kolektor (otwarty dren) stanu wyjść wszystkich urządzeń podłączonych do szyny. W przypadku jednoczesnego nadawania przez kilka urządzeń nadrzędnych kontrolę nad szyną utracą te, które będą usiłowały nadać bit danych równy jeden, podczas gdy pozostałe będą wysyłały zero. Jednoczesne nadawanie przez kilka urządzeń nadrzędnych powoduje, że sygnał taktujący przesyłany linią SCL jest również funkcją AND typu otwarty kolektor sygnałów taktujących poszczególnych urządzeń nadrzędnych. Skutkiem przejścia linii SCL w stan niski, wymuszony przez dowolne z nadających urządzeń nadrzędnych, jest jednoczesne rozpoczęcie odmierzania czasu trwania niskiego stanu linii SCL przez wszystkie nadające urządzenia nadrzędne (rys. 10.20). Powrót linii SCL do stanu wysokiego nastąpi w momencie, gdy najwolniejsze z nadających urządzeń nadrzędnych odmierzy swój okres niskiego stanu linii SCL. Narastające zbocze sygnału SCL jest chwilą synchronizacji wszystkich nadających układów nadrzędnych, po której zaczynają one odmierzać czas trwania stanu wysokiego na linii SCL. Oczywiście, ponowne przejście linii SCL w stan niski zostanie wymuszone przez najszybsze z nadających urządzeń nadrzędnych, po czym cała opisana wyżej operacja synchronizacji będzie powtarzana. Tak działająca procedura synchronizacji gwarantuje, że zarówno czas trwania niskiego jak i wysokiego stanu na linii SCL mieści się w granicach dopuszczanych przez standard szyny I²C.

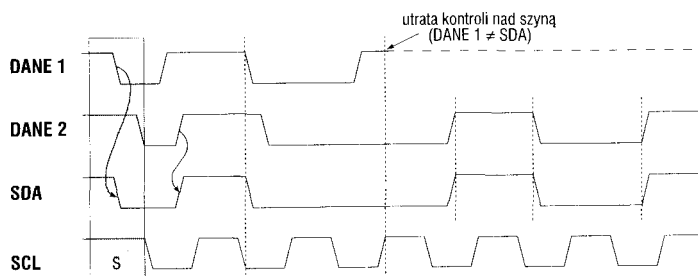
Urządzenie nadrzędne może rozpocząć nową transmisję tylko, jeśli szyna jest zwolniona. Tak więc po zakończeniu jednej transmisji kilka urządzeń nadrzędnych może jednocześnie próbować rozpocząć nową transmisję przez wysłanie sygnału START.



Rys. 10.20. Synchronizacja sygnałów taktujących podczas arbitrażu dostępu do szyny

Dzięki procedurze synchronizacji, wygenerowany sygnał START będzie spełniał warunki narzucone przez standard I²C, bez względu na szybkość nadających urządzeń – podobnie jak w przypadku sygnału taktującego. Po wysłaniu sygnału START jest uruchamiany mechanizm arbitrażu. Mechanizm ten sprawdza stan linii SDA podczas wysokiego stanu na linii SCL. W momencie gdy układ arbitrażu stwierdzi, że linia SDA znajduje się w stanie niskim (wymuszonym przez jeden z pozostałych nadających układów nadrzędnych), podczas gdy przez dany układ nadrzędny jest wysyłany stan wysoki, stopnie końcowe linii sygnałowych danego urządzenia nadrzędnego są wyłączane i układ traci kontrolę nad szyną (rys. 10.21). Arbitraż może być kontynuowany przez wiele bitów. W pierwszym rzędzie porównywane są nadawane adresy urządzeń podrzędnych. Jeśli kilka urządzeń nadrzędnych próbuje zaadresować to samo urządzenie podrzędne, mechanizm arbitrażu funkcjonuje dalej – podczas transmisji danych. W ten sposób arbitraż nie powoduje utraty żadnej informacji. Urządzenie nadrzędne, które straciło kontrolę nad szyną może wysłać impulsy taktujące do końca bajtu, podczas którego utraciło kontrolę nad szyną. Jeśli dane urządzenie może pracować także i w trybie urządzenia podrzędnego, a utrata kontroli nad szyną nastąpiła podczas adresowania urządzenia podrzędnego, to urządzenie przełącza się natychmiast (automatycznie) w tryb pracy urządzenia podrzędnego. Istnieje bowiem możliwość, że urządzenie nadrzędne, które przejęło kontrolę nad szyną usiłuje zaadresować jako podrzędne to, które przegrało właśnie arbitraż. Analizując mechanizm arbitrażu można łatwo dojść do wniosku, że w systemie wykorzystującym interfejs I²C nie występuje żaden priorytet między poszczególnymi urządzeniami nadrzędnymi, ponieważ arbitraż jest przeprowadzany zawsze wyłącznie na podstawie nadawanych adresów i danych.

Mechanizm synchronizacji sygnału taktującego, oprócz wykorzystania w procedurach arbitrażu dostępu do szyny, stanowi również formę handshake'u zarówno na poziomie bajtowym jak i bitowym poszczególnych transmisji. Jego wykorzystanie pozwala na dostosowanie prędkości transmisji do maksymalnej możliwej w danych warunkach. Problemy związane z transmisją na poziomie bajtowym polegają na tym, że urządzenie może szybko odbierać poszczególne bajty przesyłanych danych, może jednak potrzebować dużo czasu na ich dalszą obróbkę lub na przygotowanie się do odbioru następnych informacji. W takich przypadkach po odebraniu i potwierdzeniu bajtu, urządzenie podrzędne może przytrzymać linię SCL w stanie niskim, wymuszając w ten sposób na urządzeniu nadrzędnym przejście w stan oczekiwania na gotowość urządzenia podrzędnego do dalszej transmisji (sygnałizowaną zwolnieniem linii SCL). Wykorzystanie handshake'u na poziomie bitowym polega na tym, że urządzenie, które np. nie ma



Rys. 10.21. Mechanizm arbitrażu na przykładzie dwóch jednocześnie nadających układów nadrzędnych

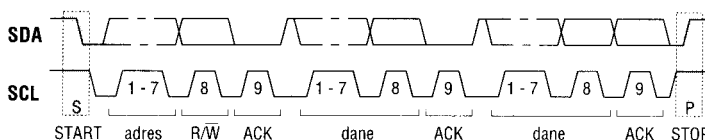
sprzętowego interfejsu I²C może zwolnić prędkość przesyłania danych przez wydłużanie czasu trwania niskiego stanu na linii SCL podczas transmisji kolejnych bitów. W takiej sytuacji urządzenie nadrzędne przystosuje prędkość transmisji do wartości wynikającej z szybkości działania urządzenia podrzędnego.

10.8.4. Protokoły transmisji danych z adresowaniem 7-bitowym

Protokół transmisji danych po szynie I²C w trybie adresowania 7-bitowego przedstawiono na rys. 10.22. Po sygnale początku transmisji (START) nadawany jest 7-bitowy adres urządzenia podrzędnego, a bezpośrednio po nim tzw. bit kierunku – określający, czy następny bajt (danych) będzie nadawany przez urządzenie nadrzędne, a odbierany przez urządzenie podrzędne (bit kierunku równy 0), czy odwrotnie (bit kierunku równy 1). Po tak przeprowadzonym adresowaniu urządzenia podrzędnego następuje transmisja bajtów danych. Przesyłanie pakietu danych (między danym urządzeniem podrzędnym i nadrzędnym) kończy się w chwili wysłania przez urządzenie nadrzędne sygnału STOP. Jeśli jednak urządzenie nadrzędne zamierza natychmiast po zakończeniu jednej transmisji rozpocząć następną, to zamiast nadawania sygnału STOP, a zaraz po nim sygnału START, może wysłać tzw. powtórny sygnał początku transmisji (Sr) i rozpocząć adresowanie kolejnego urządzenia podrzędnego (bez uprzedniego wysyłania sygnału STOP). Przykłady typowych protokołów transmisji po szynie I²C przedstawiono na rys. 10.23.

Część produkowanych układów posiadających interfejs I²C ma adres ustalony z góry – adres ten nie może być w żaden sposób zmieniony, a w konsekwencji niemożliwe jest dołączenie dwóch takich układów do szyny I²C (nie byłby spełniony wymóg unikatowych adresów urządzeń). Niektóre układy wyposażone w interfejs I²C posiadają specjalne wyprowadzenia służące do sprzętowego definiowania części ich adresu – np. dwóch najmłodszych bitów. Takie rozwiązanie umożliwia dołączenie do szyny kilku identycznych układów (np. przetworników lub pamięci EEPROM), ponieważ możliwe jest nadanie tym urządzeniom różnych adresów (przez odpowiednie wysterowanie wymienionych adresowych wyprowadzeń tych układów). Mikrokontrolery, jako układy z wbudowaną inteligencją, posiadają możliwość płynnej, programowej zmiany własnego adresu (wykorzystywanego w trybie pracy jako urządzenie podrzędne).

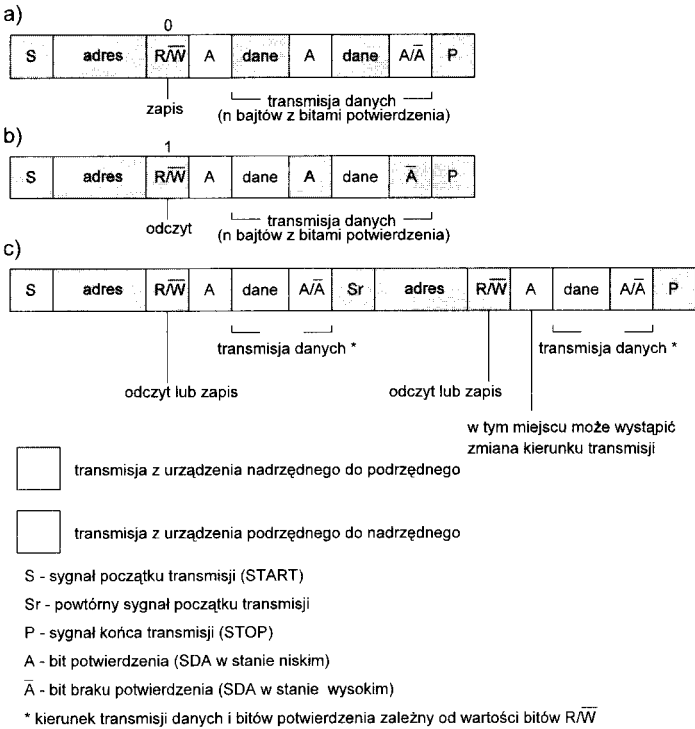
Zazwyczaj 7-bitowy adres służy do jednoznacznego zaadresowania urządzenia podrzędnego. Wyjątkiem od tej reguły jest m.in. tzw. adres wywołania ogólnego, który może zaadresować wszystkie urządzenia podrzędne podłączone do szyny (tab. 10.1). Teoretycznie w odpowiedzi na adres wywołania ogólnego wszystkie urządzenia podrzędne powinny wysłać bit potwierdzenia. W praktyce jednak niektóre z nich mogą ignorować adres wywołania ogólnego. Reakcję urządzeń na wysłanie adresu wywołania ogólnego określa zawartość drugiego bajtu, wysłanego bezpośrednio



Rys. 10.22. Protokół transmisji danych magistralą I²C w trybie adresowania 7-bitowego

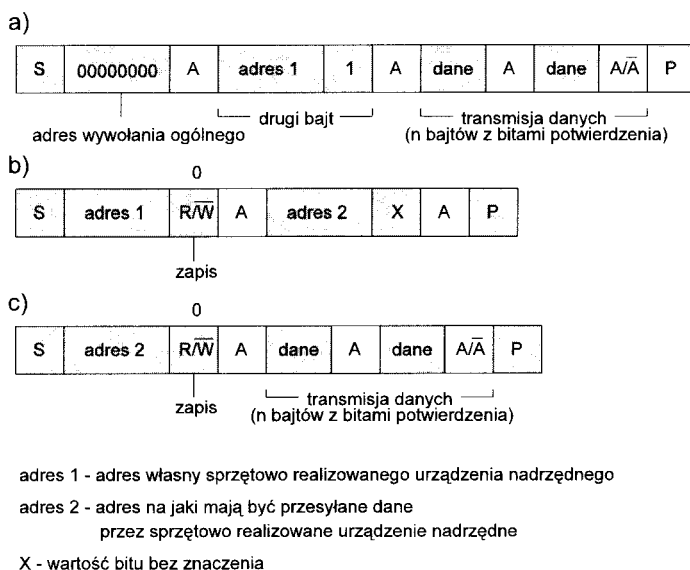
Tab. 10.1. Znaczenie pierwszego bajtu wysyłanego przez urządzenie nadrzędne

Wysłany adres urządzenia podrzędnego	Bit kierunku (R/W)	Opis
0000000	0	Adres wywołania ogólnego
0000000	1	Bajt startu
0000001	X	Adres urządzeń z interfejsem CBUS
0000010	X	Adres zarezerwowany dla urządzeń z innymi rodzajami interfejsu szeregowego
0000011	X	Zarezerwowane
00001XX	X	Zarezerwowane
11111XX	X	Zarezerwowane
11110XX	X	Pierwszy bajt 10-bitowego adresu urządzenia podrzędnego
pozostałe	X	Bajt 7-bitowego adresu urządzenia podrzędnego



Rys. 10.23. Protokół transmisji danych magistralą I²C w trybie adresowania 7-bitowego:
a) z zaadresowaniem urządzenia podrzędnego do odbioru; b) z zaadresowaniem urządzenia podrzędnego do nadawania; c) transmisja mieszana (np. najpierw nadawanie, potem odbiór danych)

po adresie wywołania ogólnego. Urządzenia, które nie są w stanie zinterpretować lub wykorzystać znaczenia tego bajtu powinny to zasygnalizować brakiem bitu potwierdzenia. Znaczenie drugiego bajtu wywołania ogólnego zależy od jego wartości. Jeśli najstarszym bitem tego bajtu jest zero, to bajt ten może mieć wartość 06h lub 04h – wszystkie inne wartości powinny być ignorowane przez odbierające je urządzenia podrzędne, ponieważ standard I²C definiuje tylko wymienione dwie kombinacje bitów drugiego bajtu wywołania ogólnego. Bajt postaci 06h powoduje wyzerowanie urządzeń podrzędnych i wczytanie przez nie tych części ich adresów, które są określane sprzętowo, natomiast kombinacja 04h powoduje jedynie wczytanie sprzętowo definiowanych fragmentów adresów, bez zerowania urządzenia. Jedynka na najstarszym bicie drugiego bajtu adresu wywołania ogólnego jest jednoznacznym wyróżnikiem tzw. sprzętowego adresu wywołania ogólnego. Siedem pozostałych bitów stanowi wówczas adres własny urządzenia wysyłającego sprzętowy adres wywołania ogólnego. Sprzętowy adres wywołania ogólnego jest generowany przez tzw. sprzętowo realizowane urządzenia nadrzędne (np. czytnik klawiatury), które nie mogą być zaprogramowane na adresowanie innych urządzeń podrzędnych (chodzi tu o wysyłanie konkretnych adresów). Wysłanie adresu wywołania ogólnego z podaniem własnego identyfikatora (adresu) jest dla takich urządzeń jedyną formą zasygnalizowania swojej obecności w systemie. Mikrokontroler (lub inne inteligentne urządzenie) odbiera taki adres i jest w stanie pokierować dalszym przepływem informacji z aktywnego sprzętowo realizowanego urządzenia nadrzędnego (rys. 10.24). Niektóre z takich urządzeń mogą też pracować jako urządzenia podrzędne i są ustawiane w taki tryb pracy np.



Rys. 10.24. Sterowanie przepływem informacji ze sprzętowo realizowanego urządzenia nadrzędnego: a) przesłanie adresu własnego podczas transmisji sterowanej przez sprzętowo realizowane urządzenie nadrzędne; b) konfigurowanie sprzętowo realizowanego urządzenia nadrzędnego (pracującego w danej chwili jako urządzenie podrzędne) przez inne urządzenie nadrzędne; c) transmisja danych sterowana przez skonfigurowane uprzednio sprzętowo realizowane urządzenie nadrzędne

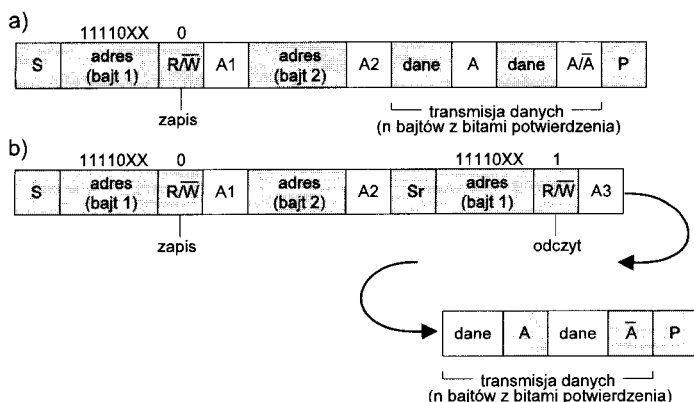
wskutek zerowania systemu. W takim przypadku urządzenie nadrzędne, które konfiguruje cały system (np. mikrokontroler) może wysłać do sprzętowo realizowanego urządzenia nadrzędnego (pracującego tymczasem jako urządzenie podrzędne) adres, pod który mają być przesyłane dane. Po wykonaniu takiej procedury skonfigurowane urządzenie może już przejść do pracy w trybie urządzenia nadrzędnego.

Mikrokontrolery wyposażone w interfejs I²C mogą być zaprogramowane tak, by wszystkie operacje związane z obsługą szyny były wykonywane na podstawie przerwań. Niektóre układy nie mają jednak sprzętowego interfejsu I²C i muszą wszystkie funkcje związane z obsługą szyny realizować programowo. W szczególności muszą one stale monitorować stan obu linii sygnałowych szyny. Oczywiście im więcej czasu mikrokontroler poświęca na obsługę interfejsu, tym mniej ma go na realizowanie pozostałych operacji, stanowiących zwykle jego podstawowe zadanie. Dodatkową istotną cechą programowej realizacji interfejsu I²C, w porównaniu do rozwiązań sprzętowych, jest niezbyt duża szybkość transmisji danych. Sposobem na częściowe zmniejszenie obciążenia mikrokontrolera programową realizacją interfejsu I²C jest stosowanie transmisji poprzedzonych specjalnym bajtem startu. W takich przypadkach każda transmisja powinna rozpoczynać się od wysłania sygnału START, po którym następuje bajt startu (tab. 10.1) postaci 00000001, bit potwierdzenia i powtórny sygnał początku transmisji (Sr). Biorąc pod uwagę, że pierwsze siedem bitów bajtu startu stanowią zera, mikrokontroler z programowo realizowanym interfejsem I²C może próbować stan linii SDA kilkakrotnie rzadziej, niż podczas transmisji bajtu adresu lub danych. Wykrycie niskiego stanu na linii SDA (czyli któregośkolwiek z zer bajtu startu) powinno spowodować przełączenie częstotliwości próbkowania linii sygnałowych interfejsu na normalną (wyższą) wartość, aby możliwe było wykrycie sygnału powtórnego początku transmisji (i wykorzystanie go do synchronizacji). W ten sposób pomiędzy kolejnymi transmisjami czas poświęcany przez mikrokontroler na obsługę interfejsu I²C ulega znacznej redukcji. Z punktu widzenia implementacji programowych interfejsów I²C istotne jest także to, że podczas nadawania bajtu startu wszystkie urządzenia dołączane do szyny muszą wysłać bit braku potwierdzenia. Impuls taktujący odpowiadający bitowi potwierdzenia nie został usunięty w tym przypadku tylko dlatego, by zachować standardowy format informacji przesyłanych po szynie. Stosowanie bajtu startu nie ma wpływu na pracę interfejsów sprzętowych, ponieważ są one zerowane w wyniku wykrycia sygnału powtórnego początku transmisji, a w związku z tym bajt startu jest przez nie ignorowany.

10.8.5. Protokoły transmisji danych z adresowaniem 10-bitowym

Z informacji zawartych w tab. 10.1 wynika, że niektóre kombinacje adresów w formacie 7-bitowym zostały zarezerwowane do celów specjalnych – przykładem może być wspomniany bajt startu. Jedną z tych kombinacji została wykorzystana do implementacji adresowania w zakresie 10 bitów. Dzięki takiemu rozwiązaniu adresowanie 10-bitowe nie ma wpływu na pracę urządzeń adresowanych przy użyciu 7 bitów. Tak więc ta sama szyna może służyć do wymiany informacji zarówno między urządzeniami posługującymi się adresowaniem 7-, jak i 10-bitowym.

W przypadku adresowania 10-bitowego, adres urządzenia jest przesyłany na dwóch bajtach. Pierwszy z tych bajtów ma postać 11110XXX, gdzie pierwsze pięć bitów



Rys. 10.25. Protokół transmisji po szynie I²C w trybie adresowania 10-bitowego: a) z zaadresowaniem urządzenia podrzędnego do odbioru; b) z zaadresowaniem urządzenia podrzędnego do nadawania

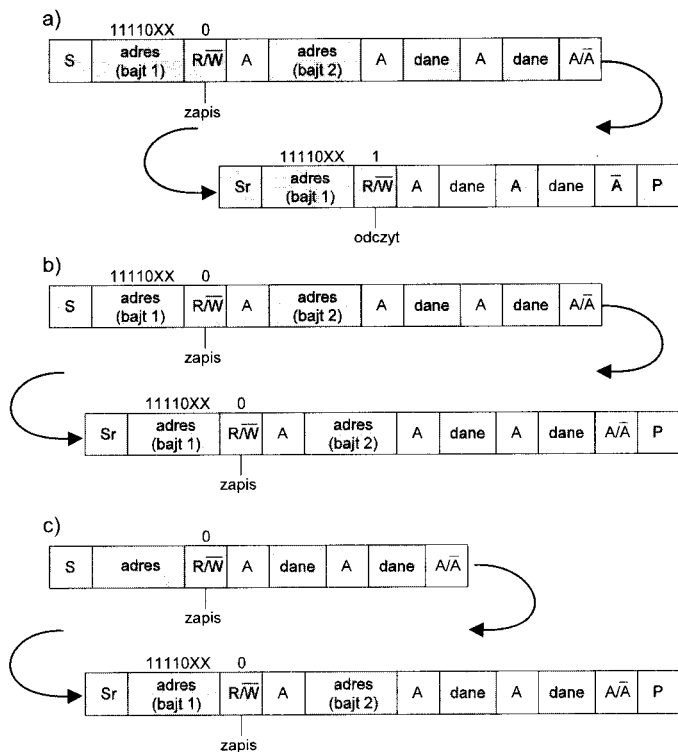
stanowi identyfikację adresowania 10-bitowego, bity XX są dwoma najstarszymi bitami adresu urządzenia podrzędnego, a bit K jest bitem kierunku, o postaci identycznej jak w adresowaniu 7-bitowym. Sposób przesyłania ośmiu młodszych bitów adresu urządzenia podrzędnego zależy od rodzaju transmisji i wynika ze sposobu dekodowania własnego 10-bitowego adresu przez układy podrzędne.

Podczas nadawania pierwszego bajtu adresu występującego po sygnale początku transmisji (START), każdy z układów podrzędnych porównuje pierwsze siedem bitów tego bajtu (ósmym jest bit kierunku) z własnym adresem. Istnieje oczywiście możliwość, że kilka układów podrzędnych stwierdzi zgodność nadanego bajtu adresu (zawierającego pierwsze dwa bity adresu 10-bitowego) z własnym adresem. W takim przypadku wszystkie te urządzenia wysłają bit potwierdzenia (A1 na rys. 10.25). Jeśli wysłany bit kierunku był zerem (co oznacza, że dane mają być nadawane przez układ nadrzędny, czyli że urządzenie podrzędne zostało zaadresowane do odbioru), urządzenie nadrzędne może wysłać kolejny bajt, zawierający osiem młodszych bitów adresu. Drugi bajtu adresu zostanie potwierdzony przez jedno już tylko urządzenie podrzędne (A2 na rys. 10.25) i właśnie to urządzenie podrzędne będzie później odbierało kolejne bajty danych nadawane przez urządzenie nadrzędne. Raz zaadresowane urządzenie podrzędne zostanie rozadresowane dopiero w wyniku wysłania przez urządzenie nadrzędne sygnału końca transmisji (STOP) lub powtórnego sygnału początku transmisji (Sr), po którym wystąpi adres innego urządzenia podrzędnego.

Procedura adresowania urządzenia podrzędnego do nadawania jest bardziej skomplikowana. Problem polega bowiem na tym, że przesłanie bitu kierunku równego jeden już w pierwszym bajcie adresu spowodowałoby natychmiast rozpoczęcie nadawania danych przez wszystkie urządzenia podrzędne, mające dwa najstarsze bity takie same jak w wysłanym bajcie adresu. Chcąc uzyskać pełne zdekodowanie adresu urządzenia podrzędnego, urządzenie nadrzędne musi najpierw wysłać dwa bajty adresu w sposób identyczny, jak podczas adresowania urządzenia podrzędnego do odbioru (czyli z wyzerowanym bitem kierunku). W ten sposób zaadresowane zostanie pojedyncze urządzenie podrzędne. Następnie urządzenie nadrzędne może wysłać powtórny sygnał

początku transmisji, a po nim ponownie nadać pierwszy bajt adresu, z tym że tym razem z właściwym już bitem kierunku. Zgodnie z tym co powiedziano wcześniej, urządzenie podrzędne nie zostanie w takim przypadku rozadresowane, ponieważ adres (pierwsze dwa bity) przesłany po powtórnym sygnale startu jest zgodny z adresem aktualnie zaadresowanego urządzenia. Zaadresowane urządzenie podrzędne wyśle zatem bit potwierdzenia (A3 na rys. 10.25), po czym rozpocznie nadawanie kolejnych bajtów danych, aż do momentu pojawienia się na szynie sygnału STOP lub powtórniego sygnału początku transmisji (z innym adresem urządzenia podrzędnego).

Znając mechanizm dekodowania adresów można oczywiście tworzyć różne mieszane formaty transmisji danych – kilka przykładów takich formatów przedstawiono na rys. 10.26. Niekiedy formaty transmisji danych mogą wykorzystywać specyficzne cechy danej konstrukcji – np. adresowanie jednobajtowe urządzenia o adresie 10-bitowym, jeśli jest ono jedynym urządzeniem 10-bitowym w całym systemie. Tworzenie specjalnych formatów opartych na specyfice struktury konkretnego systemu nie jest chyba jednak najlepszym rozwiązaniem, ponieważ w niewielkim stopniu (i nie zawsze) przyspiesza uruchamianie danego systemu i zwiększa szybkość transmisji, a bardzo często stanowi istotną przeszkodę w wykorzystaniu tych samych modułów bibliotecznych przy opracowywaniu innych urządzeń.

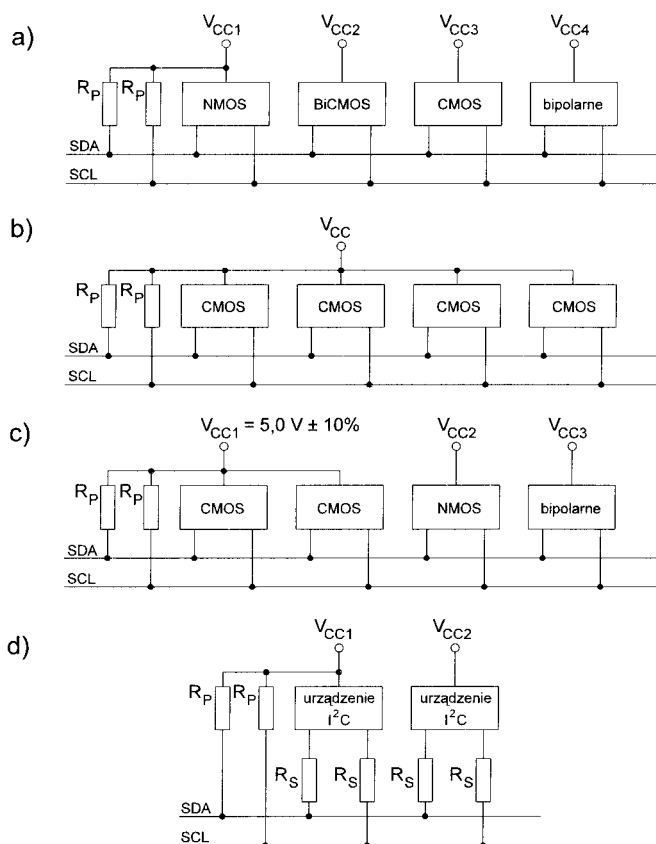


Rys. 10.26. Przykłady mieszanych formatów transmisji danych po szynie I²C: a) zapis i odczyt danych z urządzenia podrzędnego o 10-bitowym adresie; b) zapis danych do dwóch różnych urządzeń podrzędných adresowanych 10-bitowo; c) zapis danych do dwóch różnych urządzeń podrzędných, z których pierwsze wykorzystuje 7-bitowy, a drugie 10-bitowy tryb adresowania

Stosowanie adresu wywołania ogólnego w odniesieniu do urządzeń o adresowaniu 10-bitowym ma skutki analogiczne jak dla urządzeń adresowanych z wykorzystaniem 7 bitów. Urządzenia nadrzędne realizowane sprzętowo z adresem 10-bitowym będą nadawały swój adres w postaci dwóch bajtów o formacie analogicznym do stosowanego w zwykłych transmisjach. Stosowanie bajtu startu postaci 00000001 ma taki sam skutek dla urządzeń z adresem 10-bitowym, jak i 7-bitowym.

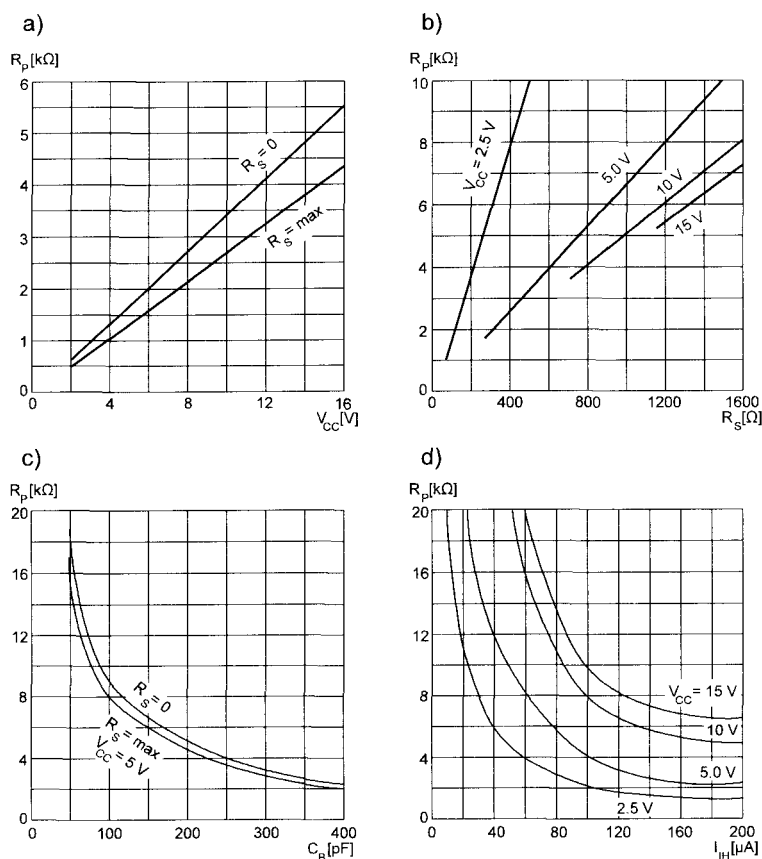
10.8.6. Elektryczne parametry szyny I²C

Parametry układów wyposażonych w interfejs I²C (tab. 10.2 i 10.3) są, w zależności od konkretnego typu układu, określone sztywno, bądź zależą od wartości napięcia zasilającego dany układ. Układy o sztywno określonych parametrach napięciowych mogą być dołączone do oddzielnych źródeł napięcia zasilającego (rys. 10.27a), natomiast wszystkie pozostałe układy peryferyjne muszą mieć jedno wspólne źródło



Rys. 10.27. Sposób dołączania zasilania i linii sygnałowych interfejsu do urządzeń z szyną I²C: a) dla układów o sztywno określonych parametrach napięciowych, b) dla układów o parametrach napięciowych zależnych od wartości napięcia zasilającego; c) w przypadku dołączania do tej samej szyny układów (a) i (b); d) z dołączeniem rezystorów R_S pełniących rolę zabezpieczenia przeciwpięciowego

napięcia zasilającego (rys. 10.27b). Jeśli do szyny dołączone są układy obu tych rodzajów, to wszystkie napięcia zasilające muszą mieć wartość $5\text{ V} \pm 10\%$ (rys. 10.27c). We wszystkich przypadkach obie linie sygnałowe interfejsu muszą być dołączone do napięcia zasilającego za pomocą rezystorów podciągających. W konstrukcjach, w których na liniach sygnałowych mogą pojawiać się zakłócenia w postaci szpilek napięciowych o znacznej amplitudzie (np. w układach odbiorników telewizyjnych), każdy z układów I²C może być zabezpieczony za pomocą rezystorów szeregowych na liniach SCL i SDA (rys. 10.27d). Wartości rezystorów podciągających R_p i rezystorów szeregowych R_s zależą od napięcia zasilania, pojemności elektrycznej linii sygnałowych oraz liczby urządzeń podłączonych do szyny (wpływ prądu wejściowego i prądu upływu). Charakterystyki umożliwiające dobranie odpowiednich rezystorów R_p i R_s w zależności od wartości wymienionych wyżej parametrów przedstawiono na rys. 10.28.



Rys. 10.28. Zależność granicznych wartości rezystancji R_s i R_p w funkcji innych parametrów elektrycznych szyny I²C: a) minimalna wartość R_p w funkcji napięcia zasilającego i rezystancji R_s ; b) zależność między maksymalną wartością R_s a rezystancją R_p przy różnych wartościach napięcia zasilającego; c) maksymalna wartość rezystancji R_p w funkcji pojemności linii sygnałowej szyny I²C dla pracy w trybie standardowym; d) całkowity prąd wejściowy w stanie wysokim w funkcji maksymalnej wartości R_p , przy różnych wartościach napięcia zasilającego

Tab. 10.2. Parametry elektryczne linii sygnałowych (SDA i SCL) interfejsu I²C

Parametr	Ozna- czenie	Praca w trybie standardowym		Praca w trybie szybkim		Jednostki
		min.	maks.	min.	maks.	
Napięcie wejściowe w stanie niskim – wartości sztywno określone – zależne od napięcia zasilania	V _{IL}	–0,5 –0,5	1,5 0,3V _{CC}	–0,5 –0,5	1,5 0,3V _{CC}	V
Napięcie wejściowe w stanie wysokim – wartości sztywno określone – zależne od napięcia zasilania	V _{IH}	3,0 0,7V _{CC}	V _{CC} +0,5 V _{CC} +0,5	3,0 0,7V _{CC}	V _{CC} +0,5 V _{CC} +0,5	V
Histereza wejść typu Schmitta – wartości sztywno określone – zależne od napięcia zasilania	V _{hys}			0,2 0,05V _{CC}	– –	V
Szerokość zakłóceń jakie muszą być odfiltrowane przez filtry wejściowe	t _{SP}			0	50	ns
Napięcie wyjściowe w stanie niskim (otwarty dren lub otwarty kolektor) – dla prądu wpływającego 3 mA – dla prądu wpływającego 6 mA	V _{OL1} V _{OL2}	0	0,4	0 0	0,4 0,6	V
Czas opadania napięcia wyjściowego od V _{IH MIN} do V _{IL MAX} przy pojemności linii sygnałowych C=10...400 [pF] – dla prądu 3 mA i V _{OL1} – dla prądu 6 mA i V _{OL2}	t _{OF}		250	20+0,1C 20+0,1C	250 250	ns
Prąd wejściowy dla napięć wejścio- wych z zakresu 0,4 [V]...0,9V _{CC}	I _I	–10	10	–10	10	μA
Pojemność wyprowadzeń	C _I		10		10	pA
Pojemność linii sygnałowe	C _B		400		400	pF

Tab. 10.3. Parametry czasowe interfejsu I²C

Parametr	Ozna- czenie	Praca w trybie standardowym		Praca w trybie szybkim		Jednostki
		min.	maks.	min.	maks.	
Częstotliwość sygnału na linii SC	f _{SCL}	0	100	0	400	kHz
Czas zwolnienia szyny (odstęp pomię- dzy sygnałami STOP i START)	t _{BUF}	4,7	–	1,3	–	μs
Odstęp między opadającymi zboczami SDA i SCL w sygnale początku trans- misji (START, Sr)	t _{HD,STA}	4,0	–	0,6	–	μs
Niski stan na linii SCL	t _{LOW}	4,7	–	1,3	–	μs
Wysoki stan na linii SCL	t _{HIGH}	4,0	–	0,6	–	μs
Odstęp między narastającym zboczem SCL i opadającym zboczem SDA w sygnale początku transmisji (START, Sr)	t _{SU,STA}	4,7	–	0,6	–	μs
Odstęp między zmianą stanu linii SDA, a narastającym zboczem SCL	t _{HD,DAT}	0	–	0	0,9	μs
Odstęp między opadającym zboczem SCL, a zmianą stanu linii SDA	t _{SU,DAT}	250	–	100	–	ns
Czas narastania sygnałów SDA i SCL	t _R	–	1000	20+0,1C	300	ns
Czas opadania sygnałów SDA i SCL	t _F	–	300	20+0,1C	300	ns
Odstęp między narastającymi zbo- czami SCL i SDA w sygnale końca transmisji (STOP)	t _{SU,STO}	4,0	–	0,6	–	μs

10.8.7. Programowa implementacja interfejsu I²C

Ze względu na dużą popularność standardu I²C, może się zdarzyć, że w projektowanym urządzeniu wygodnie byłoby zastosować układy z takim interfejsem, podczas gdy mikrokontroler nie ma wbudowanego interfejsu sprzętowego. W takich przypadkach jest możliwa programowa implementacja interfejsu I²C. Wybierając takie rozwiązanie należy mieć jednak świadomość, że w praktyce nadaje się ono tylko do dość prostych zastosowań, w których szybkość i niezawodność transmisji jest niekrytyczna, a w systemie jest wykorzystywane tylko jedno urządzenie nadrzędne. Przykłady procedur implementujących podstawowe operacje interfejsu I²C przedstawiono w list. 10.7.

List. 10.7. Przykładowe procedury programowej implementacji podstawowych operacji interfejsu I²C

```
; *****
; zestaw podstawowych procedur do programowej obsługi łącza I2C
; *****

EXTERN SDA, SCL                ; linie portów pełniące funkcje szyny I2C
EXTERN I2C_fault, I2C_busy, I2C_no_ack, I2C_ostatni_bajt
                                ; wskaźniki stanu i błędów szyny

EXTERN licznik_bitow_I2C
PUBLIC zapis_bajtu_I2C, odczyt_bajtu_I2C, start_I2C, stop_I2C
    ORG 1800H

; =====
; procedura pomocnicza - generuje opóźnienie ok. 5 us
;
; UWAGA! dopasowana do zegara 12 MHz, przy większej lub
; wyraźnie mniejszej fosc należy ją odpowiednio zmodyfikować
; =====

wait_5us:
    NOP                        ; uwzględniając, że LCALL i RET trwają
                                ; w sumie 4 cykle
    RET                        ; maszynowe wystarczy dołożyć
                                ; pojedynczą instrukcję NOP

; =====
; procedura pomocnicza - ustawia linie SCL w stan wysoki
; i czeka na zakończenie ewentualnego przeciągania stanu
; niskiego przez urządzenie podrzędne
; =====

SCL_high:
    SETB SCL                  ; próbuje wymusić wysoki stan linii SCL
    JNB SCL, $                ; i czeka na faktyczne przejście SCL
                                ; w stan wysoki
    RET

; =====
; procedura pomocnicza - wysyła sygnał STOP i zwalnia szynę
; =====

stop_I2C:
    CLR SDA                   ; wymusza stan niski na SDA
    CALL SCL_high             ; przygotowanie SCL (SCL w stan wysoki)
    CALL wait_5us             ; minimalne opóźnienie
    SETB SDA                  ; właściwe zbocze sygnału STOP
    CALL wait_5us             ;
    CLR I2C_busy               ; wyzerowanie wskaźnika zajętości szyny
    RET
```

```

; =====
; procedura wysyłania bajtu do urządzeń podrzędnych
; bajt do wysłania w ACC
;   niszczy CY
; =====

zapis_bajtu_I2C:
    MOV licznik_bitow_I2C, #8
wr_loop:
    RLC A                ; bit do wysłania do CY
    MOV SDA, C           ; i na linii SDA
    CALL SCL_high        ; zbocze narastające SCL
    CALL wait_5us        ; odmierzenie minimalnego czasu wysokiego
                        ; SCL
    CLR SCL              ; i zbocze opadające SCL
    CALL wait_5us        ; zmów minimalny odstęp czasowy
    DJNZ licznik_bitow_I2C, wr_loop
                        ; powtarza pętlę, aż wyśle wszystkie bity
    SETB SDA             ; przygotowanie do odbioru bitu
                        ; potwierdzenia
    CALL SCL_high        ; początek impulsu zegarowego dla bitu
                        ; potwierdzenia
    CALL wait_5us
    JNB SDA, no_wr_ACK    ; sprawdzenie, czy jest potwierdzenie,
                        ; jeśli nie
    SETB I2C_no_ack       ; to ustawiany jest odpowiedni znacznik
no_wr_ACK:
    CLR SCL              ; koniec impulsu zegarowego dla bitu
                        ; potwierdzenia
    CALL wait_5us
    RET

; =====
; procedura wysyłania sygnału START szyny I2C
; =====

start_I2C:
    SETB I2C_busy        ; ustawienie wskaźnika aktywnej transmisji
                        ; I2C
    CLR I2C_no_ack        ; zerowanie wskaźników błędów: braku
                        ; potwierdzenia
    CLR I2C_fault         ; oraz błędnego stanu szyny
    JNB SCL, fault        ; sprawdza poprawność stanu szyny
    JNB SDA, fault        ; i sygnalizuje ewentualny błąd
    CLR SDA               ; początek sygnału START
    CALL wait_5us
    CLR SCL
    CALL wait_5us         ; koniec sygnału START
    RET
fault:
    SETB I2C_fault        ; ustawienie wskaźnika błędu szyny
    RET

; =====
; procedura odbierania bajtu od urządzenia podrzędnego
;
;   odebrany bajt zwracany jest w ACC
;
; UWAGA! wskaźnik I2C_ostatni_bajt jest wykorzystywany do
; sygnalizacji końca odbioru danych brakiem potwierdzenia
; i musi zawierać "1", jeśli odbierany bajt ma być ostatnim
; odbieranym bajtem lub "0" w przeciwnym przypadku
;
;   niszczy CY
; =====

```

```

odczyt_bajtu_I2C:
    MOV licznik_bitow_I2C, #8          ; trzeba odebrać 8 bitów
rd_loop:
    CALL SCL_high                     ; zbocze zegara dla odbieranego bitu
    CALL wait_5us
    MOV C, SDA                        ; odbierany bit z linii SDA do CY
    RLC A                             ; i jego wsunięcie do ACC
    CLR SCL                           ; końcowe zbocze zegara odebranego bitu
    CALL wait_5us                     ; minimalny odstęp przed kolejnym bitem

    DJNZ licznik_bitow_I2C, rd_loop    ; jeśli nie wszystkie bity, to skok
    MOV C, I2C_ostatni_bajt           ; stan bitu potwierdzenia na linii SDA
    MOV SDA, C                        ; początek bitu potwierdzenia
    CALL SCL_high
    CALL wait_5us

CLR SCL
    SETB SDA                          ; koniec bitu potwierdzenia
    CALL wait_5us
    RET
END

```

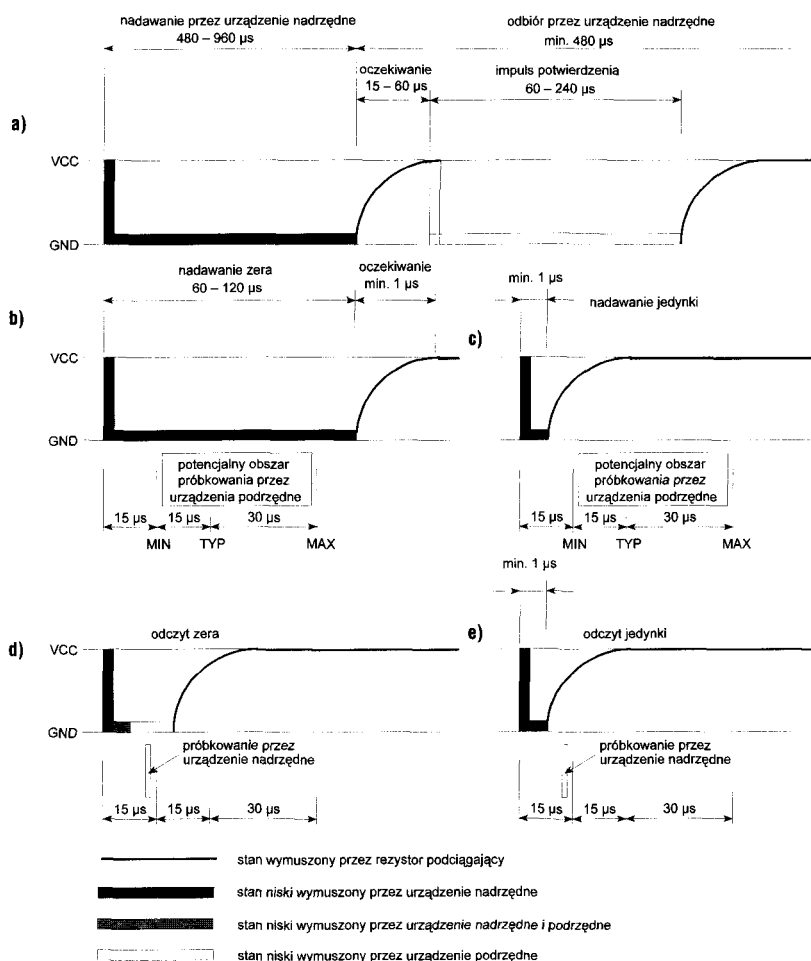
10.9. Interfejs 1-Wire

Interfejs 1-Wire jest opracowanym przez firmę Dallas Semiconductor (obecnie Maxim) interesującym rozwiązaniem interfejsu szeregowego. Interfejs ten jest przeznaczony do stosowania w systemach z pojedynczym urządzeniem nadrzędnym (*master*), sterującym wieloma urządzeniami podrzędnymi (*slave*) za pomocą jednej tylko linii (nie licząc masy), po której może być również przesyłane zasilanie układów podrzędnych. W chwili obecnej dostępnych jest wiele układów peryferyjnych z interfejsem 1-Wire. Są to m.in. czujniki temperatury z wyjściem cyfrowym, pamięci RAM i EEPROM, układy zegarów czasu rzeczywistego, potencjometry cyfrowe itp. Biorąc pod uwagę, że komunikacja po szynie 1-Wire jest precyzyjnie sterowana ze strony układu nadrzędnego (mikrokontrolera) i nie narzuca bardzo silnych uwarunkowań czasowych, procedury realizujące podstawowe operacje wymiany informacji są stosunkowo proste i warto pokusić się o ich programową implementację. W tym celu należy jednak najpierw poznać podstawy działania interfejsu.

Jak wspomniano, do wymiany informacji jest wykorzystywana pojedyncza dwukierunkowa linia sygnałowa. Żeby nie występowały konflikty wyjść, wszystkie urządzenia dołączone do szyny powinny mieć układy wyjściowe pracujące w trybie otwarty dren (lub otwarty kolektor), pozwalający im na wymuszanie stanu niskiego na magistrali, podczas gdy stan wysoki jest wynikiem działania pojedynczego rezystora podciągającego o wartości około 5 kΩ, który należy dołączyć zewnętrznie między szynę interfejsu i napięcie zasilania (typowo 3,0...5,5 V). Oczywiście przy takiej konfiguracji stan wysoki pojawi się na szynie tylko wówczas, gdy wszystkie tranzystory stopni wyjściowych urządzeń dołączonych do magistrali będą wyłączone (jest to realizacja funkcji iloczynu logicznego jako tzw. *wired AND*), a włączenie jednego (lub kilku) z tych tranzystorów powoduje natychmiast pojawienie się na szynie stanu niskiego. Sekwencje sterujące, jakie mogą pojawić się na szynie 1-Wire można w zasadzie sprowadzić do pięciu przypadków: zerowania układów podrzędnych, zapisu zera, zapisu jedynki, odczytu zera i odczytu jedynki.

Operacja zerowania układów podrzędnych służy również do sprawdzenia, czy występują jakiegokolwiek urządzenia podrzędne dołączone do magistrali. Operacja ta przebiega w ten

sposób, że mikrokontroler (jako układ nadrzędny) wymusza na szynie 1-Wire impuls ujemny o czasie trwania w zakresie 480...960 μs , na który układy podrzędne odpowiadają (po 15...60 μs od zakończenia impulsu zerującego) własnymi impulsami ujemnymi o czasie trwania 60...240 μs , potwierdzającymi ich obecność (rys. 10.29a). Parametry czasowe impulsów potwierdzających są tak dobrane, że w przypadku kilku urządzeń podrzędnych dołączonych do szyny, ich impulsy potwierdzające nakładają się na siebie – dzięki temu układ nadrzędny zawsze widzi jeden (tyle, że co najwyżej nieco dłuższy) impuls potwierdzający. W celu wyzerowania i sprawdzenia obecności układów podrzędnych wystarczy zatem, jeśli mikrokontroler wystawi na szynę 1-Wire ujemny impuls o czasie trwania np. 720 μs , a następnie odczeka jeszcze 70 μs i sprawdzi stan szyny. Odczytany wówczas stan niski będzie jednoznacznym wskaźnikiem, że do szyny są podłączone jakieś urządzenia podrzędne; stan wysoki będzie oznaczał brak takich układów.



Rys. 10.29. Przebiegi czasowe standardowych sygnałów sterujących interfejsu 1-Wire: a) zerowanie układów podrzędnych; b) zapis zera do układu podrzędnego; c) zapis jedynki do układu podrzędnego; d) odczyt zera z układu podrzędnego; e) odczyt jedynki z układu podrzędnego

Operacje zapisu zera i jedynki są jeszcze prostsze. Zapis zera polega na wymuszeniu przez mikrokontroler na szynie 1-Wire stanu niskiego na czas około 60...120 μ s (rys. 10.29b), zaś zapis jedynki – na wygenerowaniu krótkiego impulsu ujemnego o czasie trwania 1...15 μ s (rys. 10.29c). W obu przypadkach układy podrzędne próbują stan szyny po czasie 15...60 μ s od opadającego zbocza impulsu wymuszonego przez mikrokontroler (pośrednio wynika stąd zatem, że po operacji zapisu jedynki przez mikrokontroler, dowolna następna operacja nie może rozpocząć się wcześniej niż po upływie 60 μ s od opadającego zbocza impulsu sygnalizującego wspomnianą jedynkę). Operacje odczytu są również niezbyt skomplikowane. Polegają one na wymuszeniu przez mikrokontroler możliwie krótkiego impulsu ujemnego (np. o czasie trwania około 1 μ s), który jest następnie podtrzymywany przez urządzenia podrzędne, jeśli wysyłają one zero (lub nie, jeśli wysyłana jest jedynka – rys 10.29c i 10.29.d). Tak więc tuż przed upływem 15 μ s od opadającego zbocza wymuszonego przez mikrokontroler, powinien on odczytać stan magistrali – stan ten będzie odpowiadał wartości bitu wysyłanego przez urządzenia podrzędne. Przykłady procedur implementujących podstawowe operacje interfejsu 1-Wire przedstawiono w list. 10.8.

List. 10.8. Przykładowe procedury implementujące programowo podstawowe operacje interfejsu 1-Wire

```
; =====
; podstawowe procedury obsługi interfejsu 1-Wire
; udostępniane są procedury jak w deklaracji PUBLIC
; wymagane zdefiniowanie (typowo w module głównym)
; parametrów jak w EXTERN
;      niszcza ACC, B, R7, CY
; =====

PUBLIC init_1_wire, zapisz_bajt_1_wire, czytaj_bajt_1_wire, RD_wire_bit
EXTERN wire, brak_urzadzen_1_wire

; =====
; prosty przykład wykorzystania makro składającego
; się z zaledwie jednego rozkazu, ale dzięki
; odpowiednio dobranej nazwie poprawiającego
; czytelność programu
; =====

macro %delay
    DJNZ B, $                ; opóźnienie równe ok. 2*B mikrosekund
endmac

; ===== koniec makrodefinicji =====

; =====
; inicjalizacja szyny (szukanie urządzeń 1-Wire)
; niszczy B i CY
; =====

init_1_wire:
    CLR wire
    MOV B, #0                ; wyzerowanie magistrali
    %delay                   ; na ok. 520 us
    SETB wire                ; powrót do stanu wysokiego
    MOV B, #30
    %delay                   ; opóźnienie ok. 65 us
    MOV C, wire              ; odczytanie stanu magistrali
```

```

        MOV brak_urzadzen_1_wire, C      ; znacznik czy są jakieś
                                          ; urządzenia 1-Wire
        %delay                          ; opóźnienie 520 us
        RET

; =====
; odczyt pojedynczego bitu
; niszczy B i CY
; =====

RD_wire_bit:                          ; odebrany bit w CY
        CLR wire                       ; krotki impuls ujemny
        SETB wire                     ; tu koniec impulsu
        MOV B, #5                     ; łącznie z reszta instrukcji daje prawie
                                          ; 15 us

        NOP
        %delay                        ; opóźnienie owych prawie 15 us
        MOV C, wire                   ; i odczyt stanu magistrali
        MOV B, #60                    ; jeszcze dodatkowe 120 us opóźnienia
        %delay
        RET

; =====
; odczyt pojedynczego bajtu (wynik w ACC)
; niszczy R7, B i CY
; =====

czytaj_bajt_1_wire:                   ; odebrany bajt w ACC
        MOV R7, #8
czytaj_kolejny_bit:
        CALL RD_wire_bit
        RRC A
        DJNZ R7, czytaj_kolejny_bit
        RET

; =====
; zapis pojedynczego bitu
; niszczy B i CY
; =====

WR_wire_bit:                          ; bit do wysłania w CY
        MOV B, #1
        CPL C
        MOV B.5, C                   ; jeśli wysyłane „0”, to B = 33, jeśli „1”
                                          ; to B = 1
        CLR wire                     ; początek ujemnego impulsu
        %delay
        SETB wire                    ; koniec impulsu po ok. 4 us („1”) lub
                                          ; 70 us („0”)
        JNC wire_1                   ; jeśli wysłana „1”, to trzeba jeszcze
                                          ; poczekać

        RET
wire_1:
        MOV B, #30                   ; przynajmniej 60 us
        %delay
        RET

; =====
; zapis pojedynczego bajtu (bajt w ACC)
; niszczy R7, B i CY
; =====

zapisz_bajt_1_wire:
        MOV R7, #8

```

```

zapisz_kolejny_bit:
    RRC A
    CALL WR_wire_bit
    DJNZ R7, zapisz_kolejny_bit
    RET

END    ; koniec modułu

```

10.10. Zegar czasu rzeczywistego

Dość często w urządzeniach mikroprocesorowych jest wykorzystywany zegar czasu rzeczywistego (RTC – *Real Time Clock*). Zdecydowana większość publikowanych projektów wykorzystuje do tego celu dedykowane układy scalone. Warto jednak zauważyć, że mając do dyspozycji mikrokontroler taktowany generatorem kwarcowym można w prosty sposób zaimplementować funkcję zegara czasu rzeczywistego programowo. Obsługa takiego programowego zegara czasu rzeczywistego pochłania minimalne zasoby mikrokontrolera, zaś jego podstawową zaletą jest cena (zerowy koszt sprzętu), nie mówiąc o miejscu na płycie. Jednocześnie używany niekiedy przez zwolenników rozwiązania sprzętowego argument konieczności ustawiania programowego zegara za każdym razem po zaniku zasilania jest niezbyt przekonujący, biorąc pod uwagę, że mikrokontroler również może być podtrzymywany bateryjnie, podobnie jak układ scalony zegara (choć będzie oczywiście pobierał nieco więcej prądu zasilania). Przykładową procedurę programowego zegara czasu rzeczywistego przedstawiono w list. 10.9.

List. 10.9. Przykładowa procedura programowej implementacji zegara czasu rzeczywistego (RTC)

```

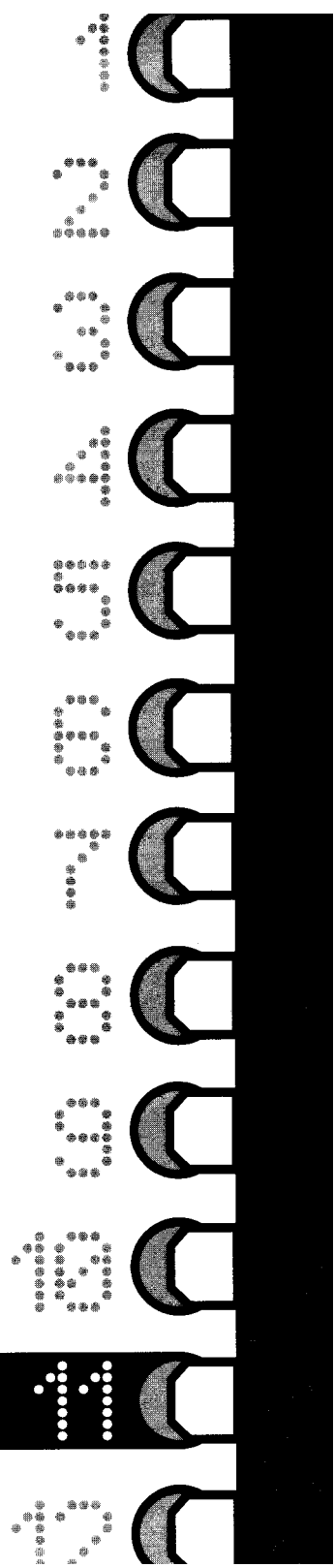
;=====
; procedura obsługi zegara czasu rzeczywistego
; przy wywołaniu w R0 musi być wskaźnik do następującej struktury:
;     TICK, SEC, MIN, HOUR
; należy zatem odpowiednio zdefiniować położenie tych zmiennych
; w module głównym i pamiętać o odpowiednim przypisaniu wartości
; do R0 przed każdym wywołaniem tej procedury!!!
;     niszczy CY
;=====

PUBLIC zegar_RTC
EXTERN f_tick                ; częstotliwość wywoływania
                                ; procedury zegara (Hz) BAJT!
EXTERN marker_sekundy        ; znacznik ustawiany każdorazowo po
                                ; odliczeniu sekundy

zegar_RTC:
    DEC @R0
    CJNE @R0, #0, return
    MOV @R0, #f_tick
    SETB marker_sekundy      ; właśnie minęła sekunda
    INC R0                   ; wskaźnik w R0 na SEC
    INC @R0
    CJNE @R0, #60, return    ; jeśli SEC <> 60, to powrót
    MOV @R0, #0
    INC R0                   ; MIN
    INC @R0
    CJNE @R0, #60, return    ; powrót, jeśli nie 60 minut

```

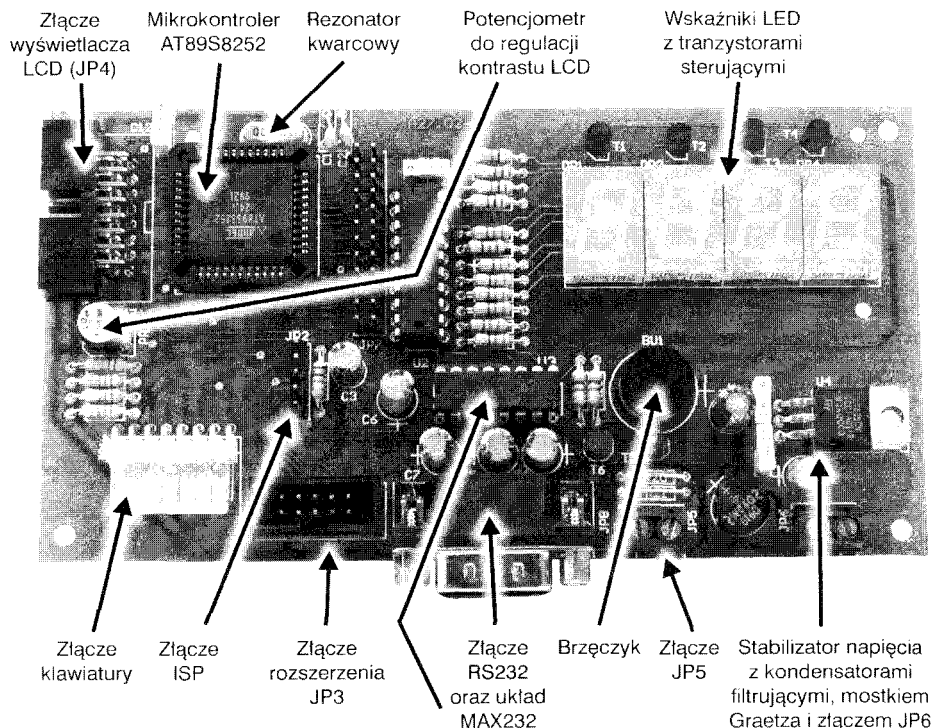

**Zestaw edukacyjny
ZL1MCS51**



Jak wspomniano w rozdz. 10 bardzo wygodnym i skutecznym sposobem nauki jest eksperymentowanie. W przypadku techniki mikroprocesorowej do tego celu bardzo często są wykorzystywane różnego typu zestawy edukacyjne. Prezentowany dalej zestaw ZL1MCS51 został opracowany głównie na potrzeby tej książki, lecz jego budowa umożliwia stosunkowo łatwe przekształcenie go w kompletne urządzenie pełniące funkcję mikroprocesorowego termometru, zegara ciemniowego, sterownika akwarium itp.

Podstawowe elementy płytki głównej zestawu to (fot. 11.1):

- mikrokontroler z pamięcią Flash U1,
- klawiatura dołączana do złącza JP1,
- konwerter poziomów RS232 ↔ TTL/CMOS (U3 z elementami towarzyszącymi),
- blok wyświetlania z 4-cyfrowym wyświetlaczem LED,
- złącze zasilania JP6,
- złącze sygnałów dodatkowych JP3 (umożliwiające bezpośrednie podłączenie do mikrokontrolera sygnałów i/lub elementów zewnętrznych – np. przetworników, czujników itp.),
- złącze interfejsu RS232 (JP8),
- złącze programatora (JP2) firmy AEC Electronics, wykorzystującego interfejs ISP (rozdz. 9.10.2),
- złącze diodowego podświetlenia wyświetlacza LCD (JP5), do którego można dołączyć także układ z optotriakiem sterujący obciążenia zasilane napięciem sieciowym lub inne elementy wykonawcze o niewielkim poborze prądu,
- złącze wyświetlacza LCD (JP4).



Fot. 11.1. Rozmieszczenie najważniejszych elementów na płycie zestawu ZL1MCS51





Płytki zestawu ZL1MCS51 można kupić w internetowym sklepie Wydawnictwa BTC, który znajduje się pod adresem www.btc.pl. Ze względu na ciągle prowadzone prace rozwojowe, w sprzedaży znajduje się udoskonalona wersja modelowego zestawu wykorzystywanego podczas opracowywania książki. Jest ona całkowicie zgodna elektrycznie z wersją prezentowaną w książce, różni się wyłącznie konstrukcją mechaniczną. Informacje o zestawie ZL1MCS51 znajdują się w Internecie pod adresem <http://www.btc.pl/index.php?id=mcs51>

Schemat elektryczny zestawu przedstawiono na rys. 11.2, a rysunki mozaiki płytek drukowanych oraz rozmieszczenie elementów w dodatku B.

Do zasilania pakietu edukacyjnego jest wymagane napięcie stałe o wartości 9...12 V lub zmienne z zakresu 9...11 V, podawane na złącze JP6. Dzięki zastosowaniu mostka B1 polaryzacja zasilającego napięcia stałego jest nieistotna. W przypadku zasilania niefiltrowanym napięciem zmiennym mostek służy jako prostownik. Właściwe napięcie zasilania (+5 V) otrzymywane jest na wyjściu stabilizatora monolitycznego 7805 (U4). Dzięki złączu JP9 stabilizowane napięcie +5 V można wykorzystać do zasilania dodatkowych układów zewnętrznych podłączanych do pakietu edukacyjnego, jednak ze względu na użycie układu 7805 bez radiatora, należy zwrócić uwagę, by prąd pobierany przez te układy nie przekraczał 50...100 mA. Złącze JP9 można też wykorzystać bezpośrednio jako wejście napięcia +5 V $\pm 10\%$ zasilającego pakiet, jednak w takim przypadku musi to być napięcie stabilizowane (otrzymywane np. ze stabilizowanego zasilacza laboratoryjnego), a jego polaryzacja jest krytyczna (jej odwrócenie może doprowadzić do nieodwracalnego uszkodzenia pakietu). Ponadto, w przypadku wykorzystania złącza JP9 jako wejścia napięcia zasilającego, nie należy montować stabilizatora U4.

Jako wyświetlacz zastosowano cztery 7-segmentowe wskaźniki LED lub, wymiennie, dowolny alfanumeryczny wyświetlacz LCD (rozstaw otworów mocujących wyświetlacz LCD do płytki głównej został zaprojektowany pod kątem modułu WM-C2002M, mającego dwa wiersze po 20 znaków). Sposób podłączenia wyświetlacza LCD umożliwia testowanie procedur obsługi zarówno dla 8-, jak i 4-bitowej szyny danych wyświetlacza. Sygnały doprowadzone do złącza wyświetlacza na płytce głównej (JP4) zostały dobrane tak, aby w przypadku wykorzystania typowego wyświetlacza LCD mającego dwurzędowe boczne wyprowadzenie sygnałów sterujących (2 \times 8 otworów z lewej strony – patrz dodatek C) wystarczyło wlutować w płytkę wyświetlacza LCD 16-końcówkowy wtyk kątowy typu BH (zamontowany od spodu wyświetlacza) i połączyć go kablem wstęgowym (tasiemką) z analogicznym wtykiem na płycie głównej. Z tego względu potencjometr PR1 do regulacji napięcia kontrastu wyświetlacza LCD znajduje się na płycie głównej. Wyświetlacz LED może być montowany bezpośrednio na płycie głównej lub na oddzielnej płytce wyświetlacza (na której złącze BH również musi być lutowane od spodu). W tym drugim przypadku do sterowania wyświetlacza LED jest wykorzystywane to samo złącze na płycie głównej i ta sama tasiemka, które służą do sterowania wyświetlaczem LCD. Wykorzystanie oddzielnej płytki wyświetlacza może być wygodne, np. gdy pakiet ma być wykorzystany jako element urządzenia zamkniętego



Jeśli wyświetlacz LED został zmontowany na płycie głównej pakietu, a ma być wykorzystywany wyświetlacz dodatkowy (LCD lub zmontowany na dodatkowej płycie wyświetlacz LED), na czas korzystania z wyświetlacza dodatkowego należy zewrzeć zworę JP7. We wszystkich pozostałych przypadkach zworę JP7 należy pozostawić rozwartą.

w obudowie, z wyświetlaczem umieszczonym na płycie czołowej. W takim przypadku nie trzeba oczywiście montować elementów wyświetlacza na płycie głównej.

Zaprojektowany wyświetlacz LED jest przystosowany do pracy dynamicznej, jednak rezystory ograniczające prąd segmentów zostały dobrane z dużym zapasem, by wskaźniki nie uległy uszkodzeniu nawet przy pracy statycznej wyświetlacza. Przy zastosowaniu wskaźników niskoprądowych do sterowania segmentów wystarczają prądy o wartości pojedynczych miliamperów, dlatego zamiast tranzystorów (jak np. na rys. 10.7) użyto układu 74HC541 (U2).

Klawiatura matrycowa 4×4 została podłączona (za pośrednictwem JP1) do mikrokontrolera w taki sposób, że możliwe jest testowanie na niej zarówno oprogramowania obsługującego klawiaturę matrycową (jak na rys. 10.4), jak i oprogramowania obsługującego pojedyncze klawisze (jak na rys. 10.3). W tym ostatnim przypadku przynajmniej na jednej z czterech starszych linii portu P0 musi być na stałe wymuszone zero (symulujące masę). W testowanym modelu wykorzystano gotową silikonową klawiaturę matrycową 4×4 firmy LC Elektronik.



Widoki mozaiki ścieżek oraz schemat ilustrujący rozmieszczenie elementów na płycie drukowanej zestawu ZL1MCS51 znajdują się w dodatku B.

Mikrokontrolerem U1 może być dowolny układ w obudowie PLCC-44 z pamięcią programu typu Flash programowalną za pomocą łącza szeregowego (interfejsu RS232) lub SPI. Obecnie (grudzień 2002 r.) układami spełniającymi ten wymóg i najłatwiej dostępnymi na naszym rynku są 89S53 i 89S8252 (programowalne za pomocą interfejsu SPI). Oba te układy są wstecznie kompatybilne z mikrokontrolerami C51 i C52, toteż jest możliwe uruchamianie na nich oprogramowania (*firmware'u*), które docelowo będzie wykorzystywane np. przez układ C51. Wartość częstotliwości rezonatora kwarcowego wynosząca 12 MHz nie jest oczywiście krytyczna (można np. wybrać popularną wartość 11,0592 MHz), jednak w przypadku wyboru innej częstotliwości i testowania załączonych procedur należy pamiętać o odpowiedniej modyfikacji stałych określających np. częstotliwość przerwań układu licznikowego T0. Jeśli rezonator kwarcowy jest w obudowie HC-49/U, zalecane jest zamontowanie go na leżąco, ponieważ przy montażu pionowym może nie zmieścić się pod płytką wyświetlacza (i spowodować zwarcie ścieżek).

Układem konwersji poziomów RS232↔TTL/CMOS jest układ MAX232 (lub jego odpowiednik). W przypadku gdy prędkość transmisji będzie niewystarczająca (maksymalna prędkość transmisji dla MAX232 wynosi ok. 60 kbd), układ ten może być zastąpiony przez MAX232A (lub jego odpowiednik) - w takim przypadku można

również zredukować pojemność kondensatorów C4, C5, C6, C7 do 100 nF. Rezystory R18, R19, R20 i R21 zostały dobrane do przykładowych zastosowań w postaci sterowania sygnalizatorem akustycznym i diodą LED, jednak w razie potrzeby sterowania z zastosowaniem większych prądów, wartości tych rezystancji można odpowiednio zmniejszyć, gdyż maksymalny prąd kolektora tranzystorów BC328 wynosi 800 mA.

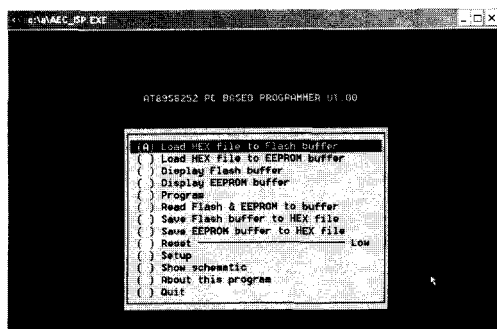
Pakiet ZL1MCS51 został przystosowany do programowania za pomocą opisanego w rozdz. 9.10.2 programatora firmy AEC Electronics (wykorzystującego interfejs SPI). Dzięki temu nakłady sprzętowe potrzebne do przeprogramowywania mikrokontrolera zostały ograniczone do przygotowania odpowiedniego kabelka, samą zaś operację programowania układu można przeprowadzać wielokrotnie, bez potrzeby brutalnego wyrywania mikrokontrolera z podstawki. Ze względu na dość skromny opis dotyczący oprogramowania programatora warto zauważyć, że kabelka programującego również nie trzeba wyciągać ze złącza po zakończeniu operacji programowania. Kod wpisany do pamięci programu można uruchomić wykonując polecenie *Reset* dostępne w menu programu (rys. 11.3). Przed rozpoczęciem jakiegokolwiek operacji programowania należy wejść w menu *Setup* i ustawić odpowiednie parametry pracy programatora (rys. 11.4).

Eksperymenty z zestawem ZL1MCS51 można zacząć od uruchomienia na nim dwóch programów testowych pokazanych w list. 11.1 oraz 11.2. Pierwszy z nich opracowano dla zestawu pracującego z alfanumerycznym wyświetlaczem LCD, demonstruje on przykładowy sposób jednoczesnej obsługi klawiatury matrycowej, wyświetlacza LCD, zegara czasu rzeczywistego i programową implementację obsługi magistrali I-Wire.

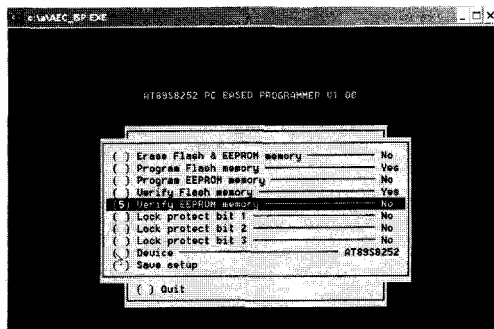
Pierwszą czynnością wykonywaną przez mikrokontroler po załączeniu zasilania zestawu jest testowanie stanu szyny I-Wire. Mikrokontroler generuje impuls zerowania układów podrzędnych, który umożliwia wykrycie ich obecności. Jeśli do szyny nie jest dołączony żaden układ I-Wire, na wyświetlaczu pojawia się komunikat „*Brak urządzeń I-Wire*”, w przeciwnym przypadku (tzn. jeśli wykryto jakiś układ I-Wire) numer identyfikacyjny tego układu jest odczytywany i wyświetlany na LCD. Po wykonaniu opisanej procedury wstępnego przeglądu szyny I-Wire, program przechodzi do nieskończonej pętli (pętla główna), a w tle (na przerwaniach od układu licznika T0) realizowane są procedury obsługi zegara czasu rzeczywistego i klawiatury. Zegar czasu rzeczywistego startuje od ustawienia 23:59:00, co umożliwia obserwację zliczania sekund i reakcji zegara na przepełnienie licznika sekund, minut i godzin. Zmiana ustawienia zegara jest możliwa za pomocą czterech dolnych klawiszy, których naciśnięcie powoduje odpowiednio zwiększanie/zmniejszanie liczby godzin/minut. Skutkiem naciśnięcia dowolnego klawisza jest również wyświetlenie jego kodu na wyświetlaczu. Żeby przetestować pełną funkcjonalność opisanego prostego programu testowego, do mikrokontrolera należy dołączyć jeden (ponieważ w programie nie zaimplementowano odczytywania numerów ID wielu urządzeń jednocześnie dołączonych do szyny) dowolny układ z interfejsem I-Wire. Można do tego celu użyć np. termometru cyfrowego DS18B20, który można będzie później wykorzystać do dalszych ekspery-



Programy prezentowane w tym rozdziale w wersjach: źródłowej i skompilowanej są dostępne w Internecie pod adresem <http://www.btc.pl/index.php?id=mcs51>.



Rys. 11.3. Główne okno programu sterującego pracą programatora AEC otwierające się po jego uruchomieniu



Rys. 11.4. Przed rozpoczęciem programowania należy skonfigurować program sterujący pracą programatora AEC

mentów – np. przekształcenia zestawu w mikroprocesorowy termometr z sygnalizacją przekroczenia zadanych temperatur progowych. Podczas podłączania układów 1-Wire do mikrokontrolera należy pamiętać, że do poprawnej pracy magistrala 1-Wire wymaga zewnętrznego rezystora podciągającego o wartości około 5 kΩ, podłączonego do napięcia zasilania (wewnętrzne rezystory podciągające stopni końcowych wyjść mikrokontrolera nie są wystarczające, gdyż mają zbyt duże rezystancje). Ponieważ w programie zadeklarowano, że linią danych 1-Wire steruje linia portu P1.4, do tej właśnie linii należy podłączyć wspomniany rezystor podciągający i wyprowadzenie DQ układu 1-Wire (lub podłączyć je do innego wolnego wyprowadzenia mikrokontrolera, jednocześnie odpowiednio modyfikując deklarację linii interfejsu 1-Wire w module głównym).

List. 11.1. Listing programu testowego dla modułu z dołączonym wyświetlaczem LCD

UWAGA

Poniżej zamieszczono jedynie listing modułu głównego programu testowego, ponieważ listingi modułów dodatkowych (obsługi wyświetlaczy, klawiatury itd.) zostały zamieszczone wcześniej – można je znaleźć w rozdziale 10.

```
; =====
; obsługa wyświetlacza LCD, klawiatury, zegara czasu rzeczywistego
; i dowolnego pojedynczego urządzenia 1-Wire dla płytki ZL1MCS51
;
;                               moduł główny
; =====
; ===   ===   ===   ===   ===   ===   ===   ===   ===   ===   ===
; deklaracje określające parametry istotne dla komunikacji
; pomiędzy modułem głównym i pozostałymi modułami

EXTERN klawiatura
PUBLIC port_klawiatury, klawisz, puszczony, wcisniety
PUBLIC licznik_klawiatury, kb_tick
EXTERN zapisz_znak_LCD, init_LCD, LCD_XY, zapisz_string_LCD
PUBLIC port_danych_LCD, E_LCD, RW_LCD, RS_LCD
EXTERN zegar_RTC
PUBLIC marker_sekundy, f_tick
EXTERN init_1_wire, zapisz_bajt_1_wire, czytaj_bajt_1_wire, RD_wire_bit
PUBLIC wire, brak_urzadzen_1_wire
```

```

; === === === === === === === === === ===
; ===== stale =====
port_klawiatury      EQU P0
f_tick               EQU 50 ; patrz opis procedury zegara RTC
kb_tick              EQU 3  ; obsługa klawiatury co 3*20 ms = 60 ms
T0_int_freq          EQU 56 ; dla 12 MHz przerwania T0 - 5kHz
port_danych_LCD      EQU P2

; ===== RAM =====
klawisz              EQU 60H
licznik_klawiatury   EQU 61H
tick                 EQU 62H
sec                  EQU 63H
min                  EQU 64H
hour                 EQU 65H
licznik_zegara       EQU 66H
temp                 EQU 67H          ; zmienna pomocnicza

; ===== bity =====
wire                 EQU P1.4          ; wyprowadzenie obsługujące
                                   ; magistrale 1-Wire
wcisniety             EQU 07FH
puszczony             EQU 07EH
marker_sekundy        EQU 07CH
brak_urzadzen_1_wire  EQU 07BH
czeka_na_puszczenie   EQU 07AH
E_LCD                 EQU P3.4; T0      ; wyprowadzenia sterujące
                                   ; wyświetlaczem LCD
RW_LCD                EQU P3.5; T1
RS_LCD                EQU P3.6; WR

;*****
; start i wektory przerwań
;*****
ORG 0000H              ; restart systemu
    JMP inicjalizacja
ORG 0BH                ; obsługa przerwania od T0
T0_int:
    DJNZ licznik_zegara, koniec_T0    ; obsługa właściwa co 20 ms
    MOV licznik_zegara, #100
    PUSH 0                        ; na stos R0 (używany jest bank
                                   ; 0 rejestrów)
    PUSH B                        ; i rejestry SFR, które mogą być
                                   ; zmienione
    PUSH ACC                      ; przez procedury wywoływane
                                   ; w przerwaniu
    PUSH PSW
    MOV R0, #tick
    CALL klawiatura                ; obsługa klawiatury
    CALL zegar_RTC                  ; i zegara czasu rzeczywistego
    POP PSW                        ; odtworzenie schowanych
                                   ; uprzednio na stos rejestrów
    POP ACC
    POP B
    POP 0
koniec_T0:
    RETI                          ; powrót z przerwania
;*****
; inicjalizacja i pętla główna programu
;*****
ORG 100H
inicjalizacja:
    MOV licznik_klawiatury, #1
    MOV licznik_zegara, #100

```

```

MOV 2FH, #0 ; zerowanie kompletu wskaźników
; bitowych
MOV sec, #0
MOV min, #59
MOV hour, #23 ; i wstępne ustawienie zegara
; RTC
; (żeby widać było jak zlicza i przepełnia się)
CALL init_LCD ; inicjalizacja wyświetlacza LCD
CALL init_1_wire ; i urządzenia 1-Wire (jeśli
; jest w systemie)
JB brak_urzadzen_1_wire, pisz_brak_1_wire
MOV DPTR, #jest_cos_1_wire ; jeśli wykryto urządzenie
; 1-Wire,
; to za chwilę będzie znane ID
CALL zapisz_string_LCD ; komenda „read ROM”
MOV A, #033H
CALL zapisz_bajt_1_wire
CALL czytaj_bajt_1_wire
CALL pisz_HEX ; ID układu dołączonego do
; 1-Wire (kod rodziny)
CALL czytaj_bajt_1_wire
CALL pisz_HEX ; i kolejne bajty
CALL czytaj_bajt_1_wire
CALL pisz_HEX
CALL czytaj_bajt_1_wire
CALL pisz_HEX
CALL czytaj_bajt_1_wire
CALL pisz_HEX
CALL czytaj_bajt_1_wire
CALL pisz_HEX
CALL czytaj_bajt_1_wire
CALL pisz_HEX
CALL czytaj_bajt_1_wire
CALL pisz_HEX ; w sumie 16 cyfr HEX
JMP dalej_init
pisz_brak_1_wire:
MOV DPTR, #pusty_1_wire ; jeśli nie wykryto urządzenia
; 1-Wire,
CALL zapisz_string_LCD ; to odpowiedni komentarz na LCD
dalej_init:
; wstępne ustawienia licznika T0 używanego przez zegar i klawiaturę
ORL TMOD, #2 ; praca T0 w trybie 2
; (z automatycznym
; przeładowywaniem)
MOV TH0, #T0_int_freq ; określa częstotl. przerwań
; od T0
ORL IE, #82H ; włączenie przerwań od
; licznika T0
SETB TR0 ; uruchomienie licznika T0
; ===== tu koniec inicjalizacji =====

main_loop: ; pętla główna
JBC marker_sekundy, nowa_sekunda ; jeśli minęła kolejna sekunda
; (skok)
JBC puszczoney, koniec_czekania_na_puszczenie
JNB wcisniety, main_loop
JB czeka_na_puszczenie, main_loop
SETB czeka_na_puszczenie ; tu właściwa obsługa klawiszy
; żeby obsługa pojedynczego
; wciśnięcia
; odbyła się jednorazowo
MOV A, klawisz ; kod klawisza do ACC
MOV A, #4EH ; kursor LCD na środek dolnego
; wiersza
CALL LCD_XY

```



```

licz:
    DIV AB                                ; po podzieleniu starsza cyfra
                                          ; w ACC

    CALL cyfra_na_ASCII
    CALL zapisz_znak_LCD
    MOV A, B                              ; młodsza w B
    CALL cyfra_na_ASCII
    CALL zapisz_znak_LCD
    RET
cyfra_na_ASCII:                          ; procedura konwersji cyfry
                                          ; (zapisanej binarnie) na kod
                                          ; ASCII

    INC A
    MOVC A, @A+PC
    RET
kody_ASCII_cyfr:
    DB '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
    DB 'A', 'B', 'C', 'D', 'E', 'F'

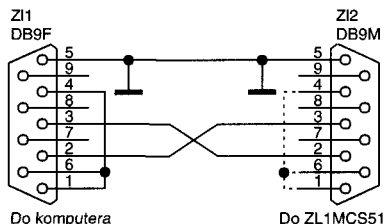
; =====
; deklaracje stringów używanych przy obsłudze 1-Wire
; =====
jest_cos_1_wire:
    DB 'ID: #'
pusty_1_wire:
    DB 'Brak urządzeń 1-Wire#'
dwie_spacje:
    DB '  #'

END                                          ; koniec modułu głównego

```

Drugi program testowy (list. 11.2) jest przeznaczony do testowania zestawu wyposażonego w wyświetlacz LED. Program ten demonstruje obsługę wyświetlacza LED sterowanego dynamicznie (multipleksowo), klawiatury i programową implementację łącza szeregowego.

Po załączeniu zasilania program wyświetla na wyświetlaczach LED napis „8051”. Jednocześnie, w tle (na przerwaniach od układu licznikowego), dokonywana jest obsługa klawiatury i programowa obsługa RS-a. Naciśnięcie dowolnego klawisza powoduje wyświetlenie kodu wciśniętego klawisza na wyświetlaczach 7-segmentowych. Cały czas jest monitorowany również stan linii danych odbieranych z interfejsu RS232 (połączenie z PC należy wykonać za pomocą kabla, którego jeden z możliwych wariantów pokazano na rys. 11.5). W przypadku odebrania jakiegokolwiek bajtu, odebrana liczba jest inkrementowana



Rys. 11.5. Sposób wykonania połączeń w kablu łączącym zestaw ZL1MCS51 z komputerem PC (DB9F oznacza gniazdo, DB9M - wtyk)

(z tym że jeśli odebrano 0FFh, czyli 255, to w wyniku inkrementacji otrzymywane jest zero) i wysyłana zwrótnie (w drugą stronę) po łączy szeregowym. Jednocześnie odebrany bajt jest wyświetlany (w postaci liczby szesnastkowej) na wyświetlaczach LED. Do testowania programowej implementacji łącza szeregowego przygotowany został specjalny program (program ten pracuje pod DOS-em, a jego skompilowana wersja EXE jest dostępna w Internecie pod adresem

<http://www.btc.pl/index.php?id=mcs51>), który umożliwia wybór portu szeregowego komputera (w zakresie COM1...COM4), do którego podłączony jest zestaw, a następnie wysyła (z szybkością około 4 bajtów na sekundę) do zestawu bajt danych (pierwszym wysyłanym bajtem jest liczba zero) i odczytuje odebrany zwrótnie bajt. Wartości obydwu bajtów (tj. wysłanego i odebranego) są wypisywane na ekranie. Odebrany bajt jest z powrotem wysyłany do zestawu. Ponieważ program testowy na mikrokontrolerze inkrementuje każdy otrzymany bajt przed jego zwróceniem, na ekranie komputera będą wyświetlane pary kolejnych liczb naturalnych (od 0 do 255), a na wyświetlaczu LED zestawu ZL1MCS51 odpowiadające im liczby szesnastkowe.

Programowa implementacja interfejsu szeregowego w programie testowym została zrealizowana na tych samych wyprowadzeniach mikrokontrolera, które wykorzystywane są przez sprzętowy interfejs mikrokontrolera. Dzięki temu można np. łatwo porównywać efekty działania obu wariantów interfejsu. Programowa implementacja interfejsu szeregowego została napisana w taki sposób, by port szeregowy można było przenieść na dowolne inne wyprowadzenia przez prostą zmianę deklaracji przypisania funkcji nadawania i odbioru do wybranych linii portu mikrokontrolera (dostępnych bitowo).

Dla ułatwienia uruchomienia stanowiska z zestawem ZL1MCS51, wraz z kodami źródłowymi programów testowych dostarczane są również skompilowane pliki HEX tych programów (w wersjach pokazanych na zamieszczonych listingach). Wstępne testowanie działania zestawu można zatem zacząć od wpisania tych plików do pamięci mikrokontrolera, a dopiero po stwierdzeniu poprawnej pracy zestawu przejść do kolejnego kroku – własnoręcznego zasemblowania i zlinkowania kompletu modułów, by uzyskać analogicznie działające pliki HEX – i wreszcie do właściwych już eksperymentów, polegających na obserwacji efektów stopniowej modyfikacji tych plików źródłowych, czy pisaniu swoich własnych procedur i programów. Należy jednak pamiętać, że w przypadku jakichkolwiek zmian w programie (czy to deklaracji, czy treści programu, oczywiście nie licząc komentarzy), cały program należy zasemblować, zlinkować i wpisać do pamięci programu mikrokontrolera. Podczas wykonywania tych operacji, ponieważ znaczna część łatwo dostępnych programów asemblerujących i łączących nie wykazuje nadmiernej „inteligencji”, należy zwrócić szczególną uwagę na odpowiednie rozmieszczenie poszczególnych modułów w przestrzeni adresowej pamięci programu. Może być w tym celu konieczne jawne użycie dyrektyw ORG z odpowiednimi wartościami adresów na początku każdego modułu. Za każdym razem (dla każdego modułu) godne polecenia jest generowanie raportów asemblacji, na podstawie których można określić jakie obszary adresowe zajmują poszczególne moduły i czy nie nakładają się one na siebie – znaczna część, zwłaszcza starszych lub freeware'owych, linkerów (programów konsolidujących) tego nie sprawdza, a nie mając wbudowanych mechanizmów automatycznej relokacji (zmiany położenia w obrębie przestrzeni adresowej), generuje pliki wynikowe kodu programu nie mające praktycznie żadnych szans na poprawne działanie.

List. 11.2. Listing programu testowego dla modułu z dołączonym wyświetlaczem LCD.



Zamieszczono jedynie listing modułu głównego programu testowego, ponieważ listingi modułów dodatkowych (obsługi wyświetlacza, klawiatury itd.) zamieszczono wcześniej – można je znaleźć w rozdziale 10.

```
; =====
; obsługa wyświetlacza LED, klawiatury
; i programowa realizacja łącza szeregowego 1200 bodów
;
;          moduł główny
; =====
; === === === === === === === === ===
;
; deklaracje określające parametry istotne dla komunikacji
; pomiędzy modułem głównym i pozostałymi modułami
EXTERN nadawanie_RS
PUBLIC zacznij_nadawanie, TX_aktywny, TX_buf, TX_pin
EXTERN odbior_RS
PUBLIC RX_aktywny, RX_buf, RX_temp, RX_pin, nr_szpilki, RX_gotowy
PUBLIC l_szpilek, polowa_l_szpilek
EXTERN klawiatura
PUBLIC port_klawiatury, klawisz, licznik_klawiatury, puszczoney
PUBLIC wcisniety, kb_tick
EXTERN wyswietlacz_LED, cyfra_na_7_seg
PUBLIC nr_LED, liczba_LEDow, anody, segmenty

; ===== stale =====
soft_baud_rate      EQU 48          ; dla 12 MHz i 1200 bodów
wygaszony           EQU 16          ; numer kodu wygaszenia w tabeli
                                ; kodów 7-seg
anody                EQU P3         ; port sterujący anodami
liczba_LEDow        EQU 4
segmenty            EQU P2
port_klawiatury     EQU P0
kb_tick             EQU 60          ; do obsługi klawiatury
l_szpilek            EQU 4          ; do obsługi programowego
                                ; RS (odbiornik)
polowa_l_szpilek    EQU 2          ; jw.

; ===== RAM =====
nr_LED              EQU 30H         ; adres bajtu z numerem
                                ; aktywnego wyświetlacza
                                ; 31 - 35H bufor wyświetlacza (+1 na podgląd klawiatury)
                                ; 36 - 39H zegar czasu rzeczywistego (tick, s, min, godz.)
klawisz             EQU 60H
licznik_klawiatury  EQU 61H
TX_aktywny          EQU 62H
TX_buf              EQU 63H
licznik_TX          EQU 64H
RX_aktywny          EQU 66H
nr_szpilki          EQU 67H
RX_temp             EQU 68H
RX_buf              EQU 69H
; ===== bity =====
wcisniety           EQU 07FH
puszczoney          EQU 07EH
zacznij_nadawanie   EQU 07DH
TX_pin              EQU P3.1
RX_pin              EQU P3.0
RX_gotowy           EQU 079H
pokazuje_8051      EQU 077H
```

```

;*****
; start i wektory przerwań
;*****
ORG 0000H                ; restart systemu
    JMP inicjalizacja
ORG 0BH                  ; przerwanie od licznika T0
T0_int:                  ; procedura obsługi przerwania od T0
    CALL odbior_RS
    DJNZ licznik_TX, koniec_TX
    MOV licznik_TX, #4
    CALL nadawanie_RS    ; obsługa nadajnika jest wywoływana
                        ; 4 x rzadziej
    CALL wyswietlacz_LED ; wywołanie procedury obsługi
                        ; wyświetlacza LED
    CALL klawiatura      ; wywołanie procedury obsługi klawiatury
    CALL pokaz_numer_klawisza
koniec_TX:
    RETI
;*****
; inicjalizacja i pętla główna programu
;*****
ORG 100H
inicjalizacja:
    MOV 2FH, #0          ; zerowanie kompletu wskaźników
                        ; bitowych
    MOV nr_LED, #1       ; inicjalizacja numeru aktywnej
                        ; anody
                        ; wyświetlacza
    ORL TMOD, #2         ; praca T0 w trybie 2
                        ; (z automatycznym
                        ; przeładowywaniem)
    MOV TH0, #soft_baud_rate ; określa częstotl. przerwań
                        ; od T0
    ORL IE, #82H         ; włączenie przerwań od
                        ; licznika T0
    SETB TR0             ; uruchomienie licznika T0
; załadowanie do bufora LED stringu „8051” (już w kodzie 7-segmentowym)
    MOV A, #8
    CALL cyfra_na_7_seg
    MOV 31H, A
    MOV A, #0
    CALL cyfra_na_7_seg
    MOV 32H, A
    MOV A, #5
    CALL cyfra_na_7_seg
    MOV 33H, A
    MOV A, #1
    CALL cyfra_na_7_seg
    MOV 34H, A
    SETB pokazuje_8051
    MOV TX_aktywny, #0
    MOV licznik_TX, #2
    MOV RX_aktywny, #0
    MOV nr_szpilki, #0
main_loop:
    JBC RX_gotowy, zwrotne_nadawanie ; jeśli odebrano coś z RS 232,
                                    ; nadać zwrotnie
    JMP main_loop                  ; jeśli nie, to nic nie rób
zwrotne_nadawanie:
    JB wciśnięty, nie_wygaszaj     ; jeśli żaden klawisz nie jest
                                    ; wciśnięty,
    MOV A, #wygaszony              ; wygasz prawe dwa wskaźniki
    CALL cyfra_na_7_seg            ; konwersja na kod 7-segmentowy
    MOV nr_LED+4, A

```

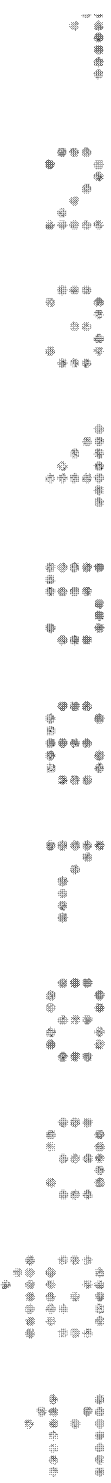
```

        MOV nr_LED+3, A                                ; (zeby już dłużej nie
                                                        ; wyświetlać „8051”)
nie_wygaszaj:
        MOV TX_buf, RX_buf                            ; odebrany przez RS bajt
                                                        ; do bufora nadawania
        INC TX_buf                                     ; i jego inkrementacja
        MOV A, RX_buf                                 ; jednocześnie odebrany bajt
        MOV B, #16
        CALL pokaz_rejestr                            ; pokazany (w HEX)
                                                        ; na wyświetlaczu LED
        SETB zacznij_nadawanie                        ; wysłanie zwrotne wartości
                                                        ; otrzymanego bajtu+1
        JMP main_loop

;=====
; procedura pomocnicza - wyświetla numer klawisza, gdy jakiś jest
; wciśnięty
;          niszczy ACC
;=====
pokaz_numer_klawisza:
        JNB wcisniety, nie_pokazuj
        JBC pokazuje_8051, wyczysc_LEDy
        SJMP pokaz_numer
wyczysc_LEDy:
        MOV A, #wygaszony
        CALL cyfra_na_7_seg                          ; konwersja na kod 7-segmentowy
        MOV nr_LED+1, A
        MOV nr_LED+2, A
        MOV nr_LED+3, A
pokaz_numer:
        MOV A, klawisz
        CALL cyfra_na_7_seg                          ; konwersja na kod 7-segmentowy
        MOV nr_LED+4, A                              ; i umieszczenie go w buforze
                                                        ; ostatniego wskaźnika
        RET
nie_pokazuj:
        JB pokazuje_8051, koniec
        MOV A, #wygaszony
        CALL cyfra_na_7_seg                          ; konwersja na kod 7-segmentowy
        MOV nr_LED+4, A                              ; i umieszczenie go w buforze
                                                        ; ostatniego
koniec:
        RET
;=====
; procedura pomocnicza - wyświetla stan rejestru
;          niszczy ACC, B, CY
;=====
pokaz_rejestr:
        DIV AB
        CALL cyfra_na_7_seg
        MOV nr_LED+1, A
        MOV A, B
        CALL cyfra_na_7_seg
        MOV nr_LED+2, A
        RET
END ; koniec modułu głównego

```

A 10x10 grid of dots where the number 12 is formed by black dots. The '1' is a vertical column of 7 dots in the 4th column from the left, with an additional dot at row 2, column 3. The '2' is formed by dots in columns 6, 7, and 8, spanning rows 2 through 9.



Literatura

- 1) Richard H. Barnett, *The 8051 Family of Microcontrollers*, Prentice Hall 1995
- 2) James W. Coffron, William E. Long, *Technika sprzęgania układów w systemach mikroprocesorowych*, WNT 1988
- 3) Jerzy Grabowski, Stanisław Koślacz, *Podstawy i praktyka programowania mikroprocesorów*, WNT 1987
- 4) William Kleitz, *Digital and Microprocessor Fundamentals. Theory and Applications*, Prentice Hall 2000
- 5) I. Scott McKenzie, *The 8051 Microcontroller*, MacMillan Publishing Company 1992
- 6) Jan Pieńkos, Stanisław Moszczyński, Adam Pluta, *Układy mikroprocesorowe 8080/8085 w modułowych systemach sterowania*, WKŁ 1988
- 7) Myke Predko, *Programming and Customizing the 8051 Microcontroller*, McGraw-Hill 1999
- 8) Tomasz Starecki, *Mikrokontrolery jednoukładowe rodziny 51*, Nozomi 1996
- 9) James W. Stewart, Kai X. Miao, *The 8051 Microcontroller. Hardware, Software and Interfacing*, Prentice Hall 1999
- 10) Ronald J. Tocci, Neal S Widmer, *Digital Systems. Principles and Applications*, Prentice Hall 2001
- 11) John Uffenbeck, *Microcomputers and Microprocessors. The 8080, 8085, and Z-80 Programming, Interfacing and Troubleshooting*, Prentice Hall 2000
- 12) Sencer Yeralan, Ashutosh Ahluwalia, *Programming and Interfacing the 8051 Microcontroller*, Addison-Wesley Publishing Company 1999
- 13) Dane katalogowe i noty aplikacyjne firm: AMD, Atmel, Cypress, Dallas Semiconductor, Intel, ISSI, Maxim, OKI, Philips, Siemens (Infineon), Winbond i innych.

Linki internetowe

Informacje o mikrokontrolerach, przykładowe projekty i kody źródłowe:

<http://www.esacademy.com/automation/docs/c51primer>
<http://www.rentron.com/8051.htm>
<http://www.boerde.de/~matthias/m8051/>
<http://www.8052.com/>
<http://www.microcontroller.com/>
<http://www4.tpgi.com.au/users/void/micro8051/>
<http://www.myke.com/>
<http://www.pjrc.com/tech/8051/>
<http://www.smartdata.com.au/8051/default.htm>
<http://www.code.archive.aisnota.com/>
http://nexus.mnic.net/~ces/ces_8051.htm
<http://microcontroller.com/embedded/references/faqs/8051-microcontroller-faq.htm>

Producenci kompilatorów dla mikrokontrolerów rodziny '51:

<http://www.archimedessoftware.com/8051.html>
<http://www.avocetsystems.com/>
<http://www.crossware.com/>
<http://www.dunfield.com/>
<http://www.fsinc.com/devtools/default.htm>
<http://www.htsoft.com/>
<http://www.hiware.com/>
<http://www.iar.com/>
<http://www.keil.com/>
<http://www.kse-softsys.com/ccomp.htm>
<http://www.mcc-us.com/>
<http://www.raisonance.com/>
<http://www.rigelcorp.com/reads51.htm>
<http://sdcc.sourceforge.net/>
<http://www.spjsystems.com/ide51.htm>
<http://www.tasking.com/products/8051/home.html>
<http://www.zetetics.com/camel/camel51.html>

Producenci mikrokontrolerów '51:

<http://www.analog.com/>

<http://www.atmel.com/>

<http://www.chipcon.com/>

<http://www.cygnal.com/>

<http://www.cypress.com/>

<http://www.goalsemi.com/>

<http://www.maxim-ic.com/>

<http://www.infineon.com/>

<http://www.issiusa.com/>

<http://www.maxim-ic.com/>

<http://www.semiconductors.philips.com/>

<http://www.smsc.com/>

<http://www.st.com/>

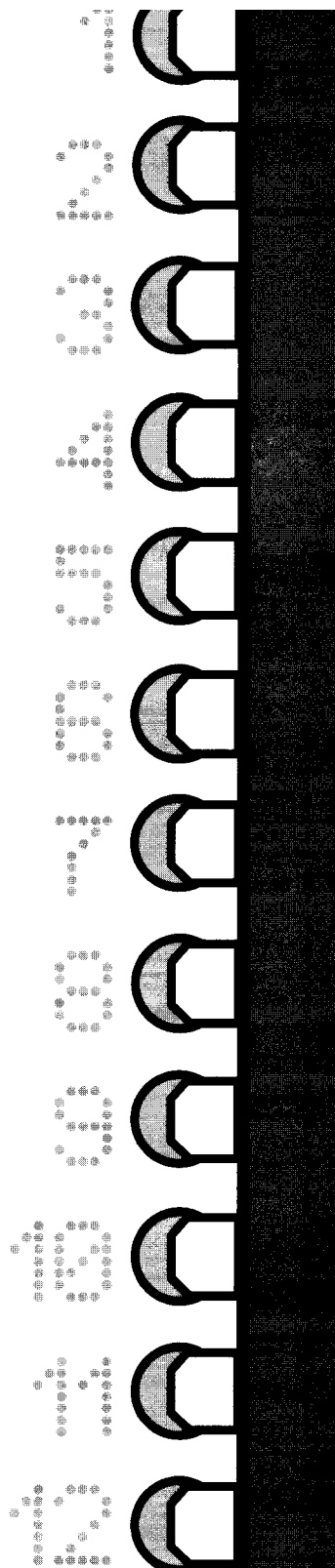
<http://www.tsc.tdk.com/>

<http://www.ti.com/>

<http://www.triscend.com/>

<http://www.winbond.com/>

Dodatki i uzupełnienia



Dodatek A. Przegląd mikrokontrolerów rodziny '51

W skład rodziny '51 wchodzi obecnie wiele dziesiątków różnych typów mikrokontrolerów. Ponieważ niniejsza książka została poświęcona podstawom projektowania układów z mikrokontrolerami rodziny '51, zostały w niej opisane jedynie najbardziej popularne typy tych mikrokontrolerów, najłatwiej dostępne na naszym rynku.

Ze względu na wyraźne zalety mikrokontrolerów wykonywanych w technologii CMOS i brak argumentów, jakie przemawiałyby za stosowaniem ich starszych odpowiedników wykonywanych w technologii NMOS, te ostatnie zostały w dalszym opisie całkowicie pominięte.

Podstawowe parametry mikrokontrolerów rodziny '51

Podstawowe parametry częściej stosowanych mikrokontrolerów rodziny '51 zostały zebrane w tabeli A.1. W dalszej części przedstawione są wybrane układy rodziny '51, z podaniem ich charakterystycznych cech.

Tabela A.1. Podstawowe parametry wybranych mikrokontrolerów rodziny '51

Wersja z pamięcią programu ROM	Wersja z pamięcią programu EPROM/Flash	Wersja z zewnętrzną pamięcią programu	Wewnętrzna pamięć programu [KB]	Wewnętrzna pamięć RAM [B]	Pamięć danych EEPROM [KB]	Liczba linii portów	Interfejs szeregowy	Interfejs SPI	Liczniki	Układy czuwające	Liczba źródeł przerwań	Liczba wyprowadzeń
80C51	87C51 89C51	80C31	4	128	—	32	1	—	2*	—	6*	40/44
—	89LV51	—	4	128	—	32	1	—	2	—	6	40/44
—	89S51	—	4	128	—	32	1	1	2	1	6	40/44
80C52	87C52 89C52	80C32	8	256	—	32	1	—	3	—	8	40/44
—	89LV52	—	8	256	—	32	1	—	3	—	8	40/44
—	89S52	—	8	256	—	32	1	1	3	1	8	40/44
—	89LS53	—	8	256	—	32	1	1	3	1	9	40/44
—	89S53	—	8	256	—	32	1	1	3	1	9	40/44
80C54	87C54 89C54	—	16	256	—	32	1	—	3	—	8	40/44
—	89C55	—	20	256	—	32	1	—	3	—	8	40/44
80C58	87C58 89C58	—	32	256	—	32	1	—	3	—	8	40/44
—	89C1051	—	1	64	—	15	—	—	1	—	3	20
—	89C1051U	—	1	64	—	15	1	—	2	—	6	20
—	89C2051	—	2	128	—	15	1	—	2	—	6	20
—	89C4051	—	4	128	—	15	1	—	2	—	6	20
—	89LS8252	—	8	256	2	32	1	1	3	1	9	40/44
—	89S8252	—	8	256	2	32	1	1	3	1	9	40/44

* najnowsze wersje mikrokontrolera C51 firmy Philips mają wbudowane 3 liczniki i 8 źródeł przerwań

80C51 (87C51, 89C51, 80C31)

Wykonany w technologii CMOS odpowiednik mikrokontrolera 8051 stanowiącego bazowy układ całej rodziny '51. Cechy charakterystyczne mikrokontrolera:

- 4 KB wewnętrznej pamięci programu typu ROM (80C51)
- 4 KB wewnętrznej pamięci programu typu EPROM (87C51)
- 4 KB wewnętrznej pamięci programu typu Flash (89C51)
- brak wewnętrznej pamięci programu (80C31)
- 128 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- dwa 16-bitowe układy licznikowe
- interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 6 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- częstotliwość taktowania od 0 Hz (Atmel, Philips, OKI)
- możliwość blokowania sygnału ALE (Atmel)

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz, inne z napięciem zasilania obniżonym nawet do 2,5 V. Najnowsze układy C51 firmy Philips mają dodatkowo układ licznikowy T2, rozszerzony interfejs szeregowy, asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera, czteropoziomowy układ przerwań i dwa wskaźniki DPTR.

89LV51

Produkowana przez firmę Atmel odmiana mikrokontrolera C51, przystosowana do pracy z niskim napięciem zasilania. Cechy charakterystyczne mikrokontrolera:

- 4 KB wewnętrznej pamięci programu typu Flash
- 128 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- dwa 16-bitowe układy licznikowe
- interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 6 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- napięcie zasilania 2,7...6,0 V
- częstotliwość taktowania od 0 Hz
- możliwość blokowania sygnału ALE

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Przy napięciu zasilania 2,7 V producent gwarantuje poprawną pracę z częstotliwością taktowania 12 MHz.

89S51

Produkowana przez firmę Atmel wersja mikrokontrolera C51 z interfejsem SPI do programowania pamięci programu. Cechy charakterystyczne mikrokontrolera:

- 4 KB wewnętrznej pamięci programu typu Flash
- 128 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- dwa 16-bitowe układy licznikowe
- interfejs szeregowy
- interfejs SPI do programowania pamięci programu
- dwupoziomowy układ przerwań, wykorzystujący 6 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- wskaźnik spadku napięcia zasilającego
- licznik czuwający
- dwa wskaźniki DPTR
- częstotliwość taktowania od 0 Hz
- możliwość blokowania sygnału ALE

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz, przy napięciu zasilania od 4,0 do 5,5 V.

80C52 (87C52, 89C52, 80C32)

Rozbudowana wersja mikrokontrolera 80C51. Cechy charakterystyczne mikrokontrolera:

- 8 KB wewnętrznej pamięci programu typu ROM (80C52)
- 8 KB wewnętrznej pamięci programu typu EPROM (87C52)
- 8 KB wewnętrznej pamięci programu typu Flash (89C52)
- brak wewnętrznej pamięci programu (80C32)
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego (m.in. Atmel, nowsze wersje Philipsa)
- rozszerzone (Intel, nowsze wersje Philipsa) interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 8 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania (Intel, nowsze wersje Philipsa)
- wskaźnik spadku napięcia zasilającego (Intel, nowsze wersje Philipsa)
- częstotliwość taktowania od 0 Hz (Atmel, nowsze wersje Philipsa)
- możliwość blokowania sygnału ALE (Atmel)

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz (Philips, Atmel). Najnowsze układy C51 firmy Philips posiadają asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera, czteropoziomowy układ przerwań i dwa wskaźniki DPTR.

89LV52

Produkowana przez firmę Atmel odmiana mikrokontrolera C52, przystosowana do pracy z niskim napięciem zasilania. Cechy charakterystyczne mikrokontrolera:

- 8 KB wewnętrznej pamięci programu typu Flash
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego
- interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 8 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- napięcie zasilania 2,7...6,0 V
- częstotliwość taktowania od 0 Hz
- możliwość blokowania sygnału ALE

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Przy napięciu zasilania 2,7 V producent gwarantuje poprawną pracę z częstotliwością taktowania 12 MHz.

89S52

Produkowana przez firmę Atmel wersja mikrokontrolera C52 z interfejsem SPI do programowania pamięci programu. Cechy charakterystyczne mikrokontrolera:

- 8 KB wewnętrznej pamięci programu typu Flash
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- interfejs szeregowy
- interfejs SPI do programowania pamięci programu
- dwupoziomowy układ przerwań, wykorzystujący 8 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- wskaźnik spadku napięcia zasilającego
- licznik czuwający
- częstotliwość taktowania od 0 Hz
- możliwość blokowania sygnału ALE
- dwa wskaźniki DPTR

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz, przy napięciu zasilania od 4,0 do 5,5 V.

89S53

Produkowana przez firmę Atmel rozbudowana wersja mikrokontrolera C52. Cechy charakterystyczne mikrokontrolera:

- 12 KB wewnętrznej pamięci programu typu Flash
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego
- interfejs szeregowy
- interfejs SPI
- dwupoziomowy układ przerwań, wykorzystujący 9 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- częstotliwość taktowania od 0 Hz
- licznik czuwający
- wskaźnik spadku napięcia zasilającego
- możliwość blokowania sygnału ALE
- dwa wskaźniki DPTR

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz.

89LS53

Produkowana przez firmę Atmel niskonapięciowa wersja mikrokontrolera S53. Cechy charakterystyczne mikrokontrolera:

- 12 KB wewnętrznej pamięci programu typu Flash
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego
- interfejs szeregowy
- interfejs SPI
- dwupoziomowy układ przerwań, wykorzystujący 9 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- częstotliwość taktowania od 0 Hz
- licznik czuwający
- wskaźnik spadku napięcia zasilającego

- napięcie zasilania 2,7...6,0 V
- możliwość blokowania sygnału ALE
- dwa wskaźniki DPTR

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 12 MHz, przy napięciu zasilania od 2,7 V.

80C54 (87C54, 89C54)

Rozbudowana wersja mikrokontrolera 80C52. Cechy charakterystyczne mikrokontrolera:

- 16 KB wewnętrznej pamięci programu typu ROM (80C54)
- 16 KB wewnętrznej pamięci programu typu EPROM (87C54)
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego (Intel)
- rozszerzony interfejs szeregowy
- czteropoziomowy (Intel, nowsze układy Philipsa) układ przerwań, wykorzystujący 8 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- możliwość blokowania sygnału ALE (Intel, nowsze układy Philipsa)
- wskaźnik spadku napięcia zasilającego
- dwa wskaźniki DPTR (nowsze układy Philipsa)
- asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz. W przypadku pracy wyłącznie z zewnętrzną pamięcią programu firma Intel zaleca stosowanie układu C32. Należy jednak pamiętać, że większość układów C32 nie realizuje 4-poziomowego układu przerwań i asynchronicznego ustawiania linii portów w wyniku zerowania mikrokontrolera, a znaczna część także blokowania sygnału ALE, i funkcji wyjścia sygnału taktującego.

89C55

Produkowana przez firmę Atmel, rozbudowana wersja mikrokontrolera 80C52. Cechy charakterystyczne mikrokontrolera:

- 20 KB wewnętrznej pamięci programu typu Flash
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego

- interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 8 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość blokowania sygnału ALE
- częstotliwość taktowania od 0 Hz
- asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz.

80C58 (87C58, 89C58)

Produkowana m.in. przez firmę Intel i Philips, wersja mikrokontrolera 80C54 ze zwiększoną pojemnością wewnętrznej pamięci programu. Cechy charakterystyczne mikrokontrolera:

- 32 KB wewnętrznej pamięci programu typu ROM (80C58)
- 32 KB wewnętrznej pamięci programu typu EPROM (87C58)
- 256 bajtów wewnętrznej pamięci danych (RAM)
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego
- rozszerzony interfejs szeregowy
- czteropoziomowy układ przerwań, wykorzystujący 8 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- możliwość blokowania sygnału ALE
- wskaźnik spadku napięcia zasilającego
- asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 33 MHz. W przypadku pracy wyłącznie z zewnętrzną pamięcią programu firma Intel zaleca stosowanie układu C32. Należy jednak pamiętać, że większość układów C32 nie realizuje 4-poziomowego układu przerwań, a znaczna część także blokowania sygnału ALE, i funkcji wyjścia sygnału taktującego.

89C1051

Produkowana przez firmę Atmel wersja mikrokontrolera 80C51 ze zredukowaną liczbą wyprowadzeń. Cechy charakterystyczne mikrokontrolera:

- 1 KB wewnętrznej pamięci programu typu Flash
- 64 bajty wewnętrznej pamięci danych (RAM)
- brak możliwości korzystania z zewnętrznej pamięci programu
- brak możliwości korzystania z zewnętrznej pamięci danych
- 15 linii wejść/wyjść
- jeden 16 - bitowy układ licznikowy
- komparator analogowy

- dwupoziomowy układ przerwań, wykorzystujący 3 źródła przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- częstotliwość taktowania od 0 Hz
- asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera

Układ jest produkowany w obudowach DIP-20, SOP-20. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 24 MHz inne z napięciem zasilania obniżonym nawet do 2,7 V.

89C1051U

Produkowana przez firmę Atmel wersja mikrokontrolera 89C1051 z dwoma układami licznikowymi i interfejsem szeregowym. Cechy charakterystyczne mikrokontrolera:

- 1 KB wewnętrznej pamięci programu typu Flash
- 64 bajty wewnętrznej pamięci danych (RAM)
- brak możliwości korzystania z zewnętrznej pamięci programu
- brak możliwości korzystania z zewnętrznej pamięci danych
- 15 linii wejść/wyjść
- dwa 16-bitowe układy licznikowe
- komparator analogowy
- interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 6 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- częstotliwość taktowania od 0 Hz
- asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera

Układ jest produkowany w obudowach DIP-20, SOP-20. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 24 MHz inne z napięciem zasilania obniżonym nawet do 2,7 V.

89C2051

Produkowana przez firmę Atmel wersja mikrokontrolera 80C51 ze zredukowaną liczbą wyprowadzeń. Cechy charakterystyczne mikrokontrolera:

- 2 KB wewnętrznej pamięci programu typu Flash
- 128 bajtów wewnętrznej pamięci danych (RAM)
- brak możliwości korzystania z zewnętrznej pamięci programu
- brak możliwości korzystania z zewnętrznej pamięci danych
- 15 linii wejść/wyjść
- dwa 16-bitowe układy licznikowe
- komparator analogowy
- interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 6 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- częstotliwość taktowania od 0 Hz
- asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera

Układ jest produkowany w obudowach DIP-20, SOP-20. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 24 MHz inne z napięciem zasilania obniżonym nawet do 2,7 V.

89C4051

Produkowana przez firmę Atmel wersja mikrokontrolera 80C51 ze zredukowaną liczbą wyprowadzeń. Cechy charakterystyczne mikrokontrolera:

- 4 KB wewnętrznej pamięci programu typu Flash
- 128 bajtów wewnętrznej pamięci danych (RAM)
- brak możliwości korzystania z zewnętrznej pamięci programu
- brak możliwości korzystania z zewnętrznej pamięci danych
- 15 linii wejść/wyjść
- dwa 16-bitowe układy licznikowe
- komparator analogowy
- interfejs szeregowy
- dwupoziomowy układ przerwań, wykorzystujący 6 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- częstotliwość taktowania od 0 Hz
- asynchroniczne ustawianie linii portów w wyniku zerowania mikrokontrolera

Układ jest produkowany w obudowach DIP-20, SOP-20. Niektóre wersje umożliwiają pracę z częstotliwością zegarową do 24 MHz inne z napięciem zasilania obniżonym nawet do 3,0 V.

89S8252

Produkowana przez firmę Atmel rozbudowana wersja mikrokontrolera C52. Cechy charakterystyczne mikrokontrolera:

- 8 KB wewnętrznej pamięci programu typu Flash
- 256 bajtów wewnętrznej pamięci danych (RAM)
- 2 KB wewnętrznej pamięci danych typu EEPROM
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego
- interfejs szeregowy
- interfejs SPI
- dwupoziomowy układ przerwań, wykorzystujący 9 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- częstotliwość taktowania od 0 Hz
- licznik czuwający
- wskaźnik spadku napięcia zasilającego
- możliwość blokowania sygnału ALE
- dwa wskaźniki DPTR

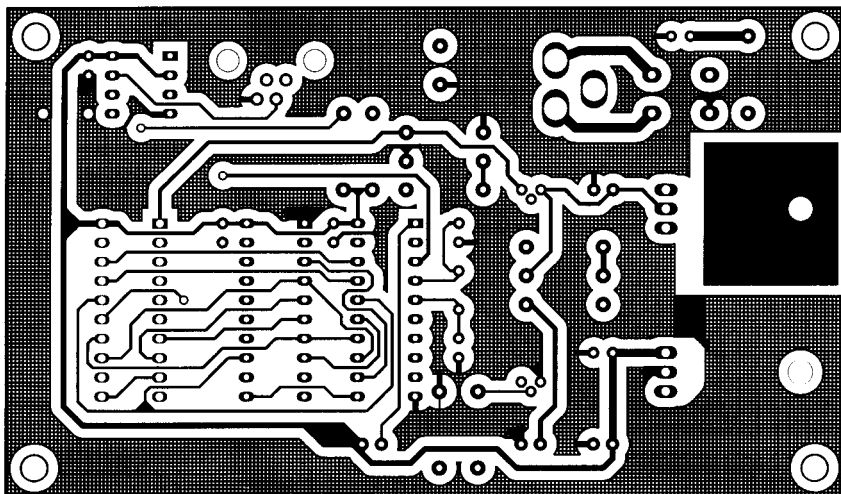
Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Niektóre wersje umożliwiają z częstotliwością zegarową do 33 MHz.

89LS8252

Produkowana przez firmę Atmel wersja mikrokontrolera S8252, przystosowana do pracy z obniżonym napięciem zasilania. Cechy charakterystyczne mikrokontrolera:

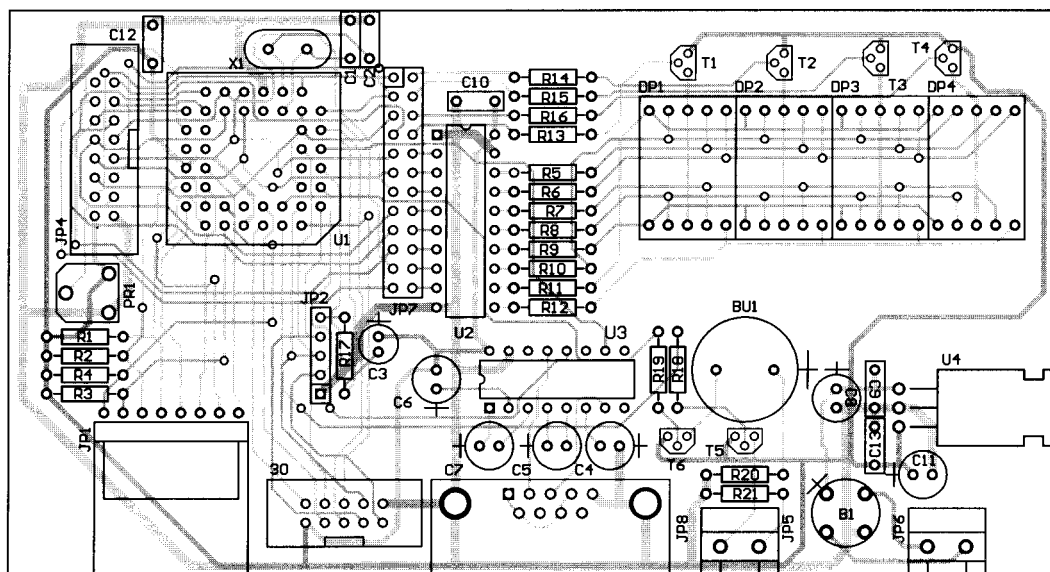
- 8 KB wewnętrznej pamięci programu typu Flash
- 256 bajtów wewnętrznej pamięci danych (RAM)
- 2 KB wewnętrznej pamięci danych typu EEPROM
- przestrzeń adresowa pamięci programu - 64 KB
- przestrzeń adresowa zewnętrznej pamięci danych - 64 KB
- 32 linie wejść/wyjść
- trzy 16-bitowe układy licznikowe
- funkcja wyjścia sygnału taktującego
- interfejs szeregowy
- interfejs SPI
- dwupoziomowy układ przerwań, wykorzystujący 9 źródeł przerwań
- tryby pracy z redukcją poboru mocy (uśpienie, zamrożenie)
- możliwość wyjścia ze stanu zamrożenia w wyniku przerwania
- częstotliwość taktowania od 0 Hz
- licznik czuwający
- wskaźnik spadku napięcia zasilającego
- napięcie zasilania 2,7...6,0 V
- możliwość blokowania sygnału ALE
- dwa wskaźniki DPTR

Układ jest produkowany w obudowach DIP-40, LCC-44, QFP-44. Układ umożliwia pracę z częstotliwością zegarową do 12 MHz przy napięciu zasilania obniżonym do 2,7 V.

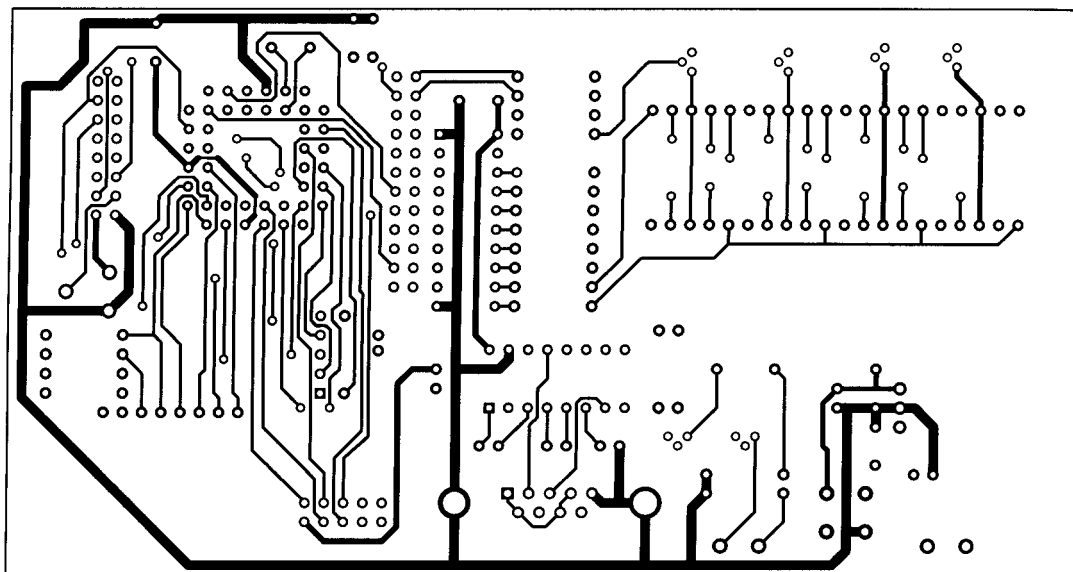


Rys. B.3. Widok strony elementów płytki programatora Easy Downloader

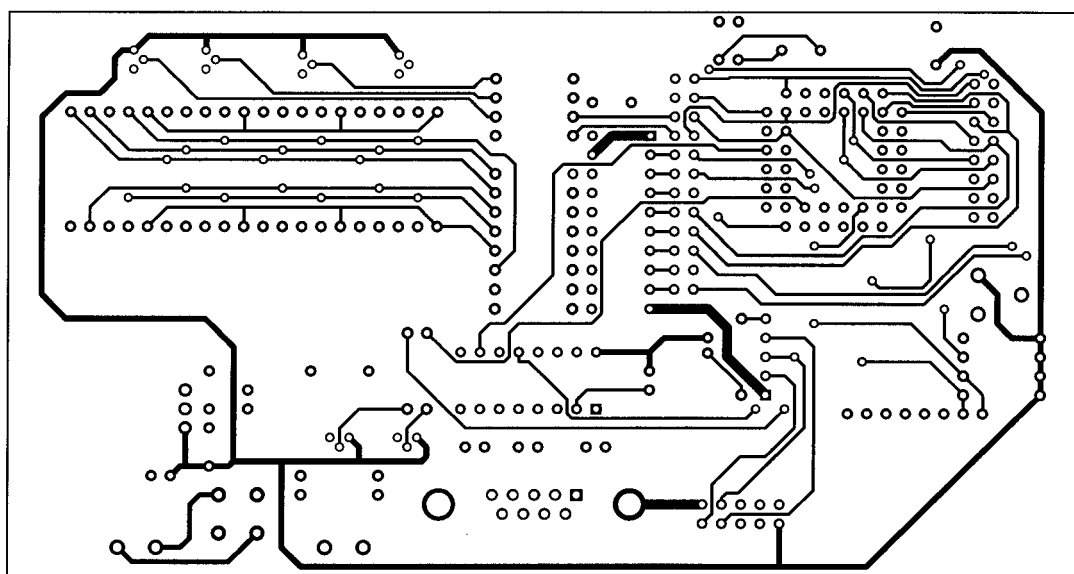
B.2. Płytką drukowaną zestawu ZL1MCS51



Rys. B.4. Rozmieszczenie elementów na płytce drukowanej zestawu ZL1MCS51

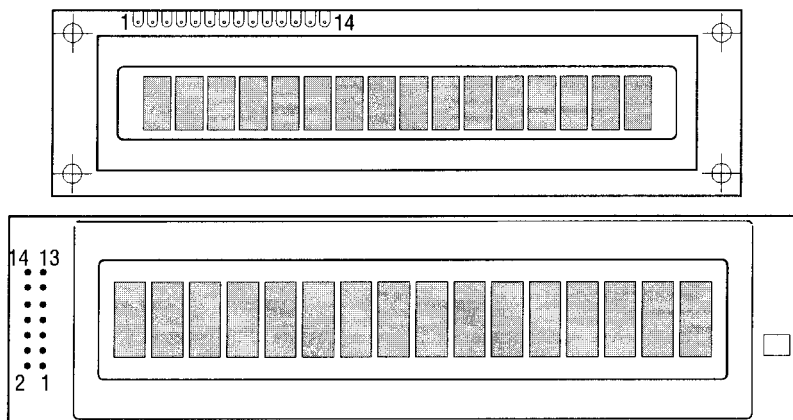


Rys. B.5. Widok strony lutowania płytki zestawu ZL1MCS51



Rys. B.6. Widok strony elementów płytki zestawu ZL1MCS51

Dodatek C. Wyprowadzenia typowych alfanumerycznych wyświetlaczy LCD



Wypr.	Symbol	Funkcja
1	VSS	Masa
2	VDD	Zasilanie +5 V
3	VO/VEE	Regulacja kontrastu
4	RS	Wybór rejestru
5	R/W	H: odczyt/L: zapis
6	E	Sygnał zezwalający (<i>enable</i>)
7	D0	Linia danych D0
8	D1	Linia danych D1
9	D2	Linia danych D2
10	D3	Linia danych D3
11	D4	Linia danych D4
12	D5	Linia danych D5
13	D6	Linia danych D6
14	D7	Linia danych D7



Rozmieszczenie wyprowadzeń modułów wyświetlaczy należy każdorazowo weryfikować, ponieważ w niektórych przypadkach może ono być inne niż pokazane na rysunku. Zalecane jest korzystanie z not katalogowych producentów wyświetlaczy, a jeżeli nie są one dostępne, rozmieszczenie wyprowadzeń można zweryfikować przez analizę położenia wyprowadzeń zasilania (masa jest najczęściej wykonana jako płaszczyzna), czasami numery skrajnych wyprowadzeń są opisane na płytce.

Dodatek D. Rozmieszczenie wyprowadzeń wybranych typów mikrokontrolerów

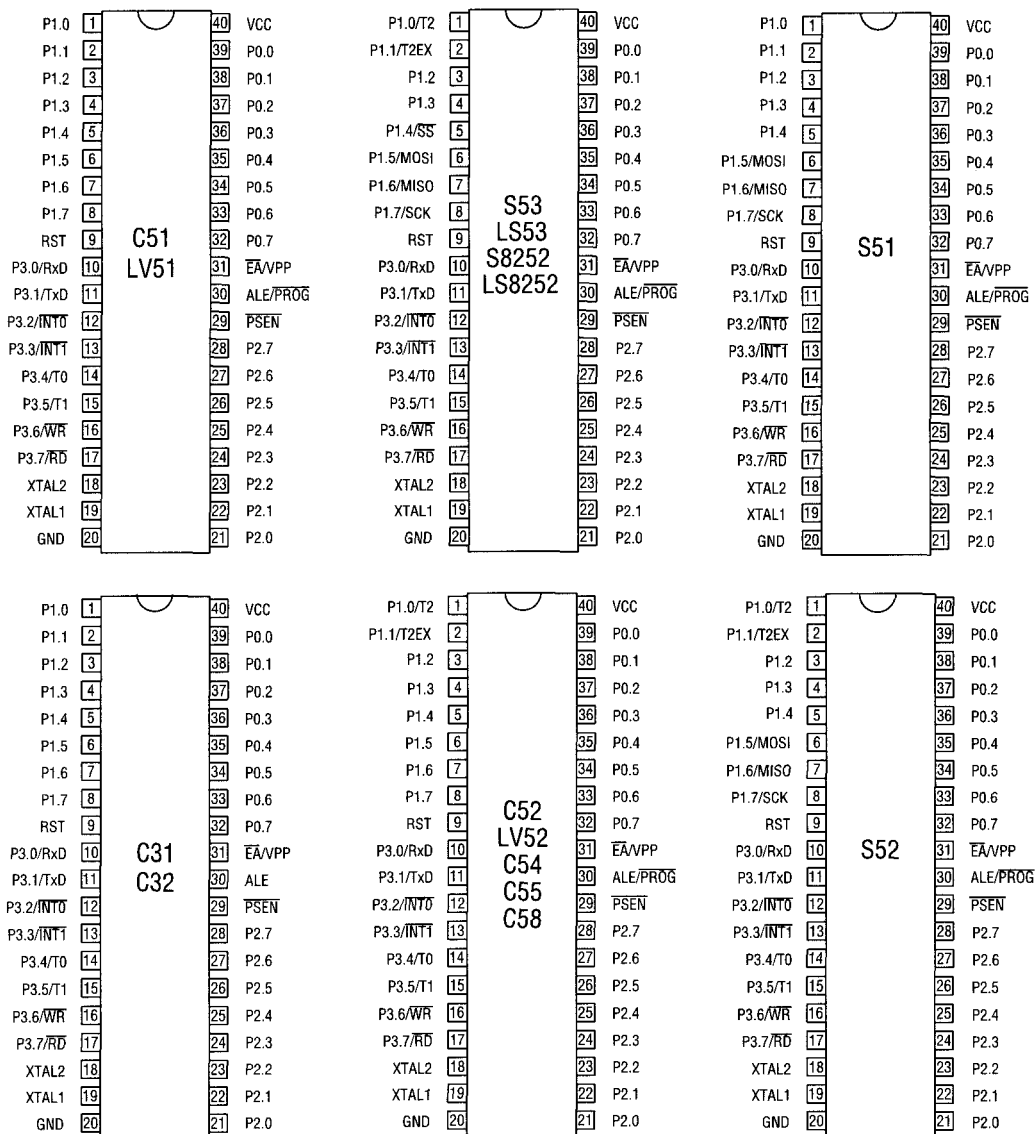
Rozmieszczenie wyprowadzeń stosowanych układów rzadko miawa wpływ na ostateczną konstrukcję (w sensie rozwiązań układowych) projektowanego urządzenia, choć nie ulega wątpliwości, że jest to informacja niezbędna do wykonania kompletnego opracowania konstrukcyjnego (przynajmniej ze względu na konieczność zaprojektowania płytki drukowanej). Mikrokontrolery są jednak dość specyficznym rodzajem układów scalonych – ze względu na możliwość pełnienia przez niektóre ich wyprowadzenia wielu różnych funkcji alternatywnych. Tak więc, istotną rolą zamieszczonych dalej rysunków, przedstawiających rozmieszczenie wyprowadzeń wybranych mikrokontrolerów rodziny '51, jest unaocznienie Czytelnikowi, że wprowadzając dany układ może mieć np. 32 linie wejść/wyjść cyfrowych, ale wykorzystywanie np. łącza szeregowego zmniejsza do 30 liczbę linii wejść/wyjść, które mogą być użyte do celów innych niż dwukierunkowa transmisja szeregową. Analizując rozmieszczenie wyprowadzeń poszczególnych mikrokontrolerów rodziny '51 należy pamiętać, że konkretne linie portów pełnią te same funkcje alternatywne bez względu na rodzaj obudowy i numer wyprowadzenia przyporządkowany danej linii. Na przykład jeśli wyprowadzenie 11 mikrokontrolera C51 w obudowie DIP-40 jest linią P3.1 pełniącą funkcję alternatywną typu TxD, to w przypadku takiego samego mikrokontrolera w obudowie PLCC-44 funkcję TxD będzie realizować również linia P3.1, tyle że będąca wyprowadzeniem o numerze 13.

Wybór rodzaju obudowy mikrokontrolera, który ma być zastosowany w projektowanym układzie zależy zwykle od uwarunkowań technologicznych – nie ma np. sensu stosowanie mikrokontrolera w obudowie DIP w urządzeniu, od którego wymaga się przede wszystkim jak najmniejszych gabarytów.

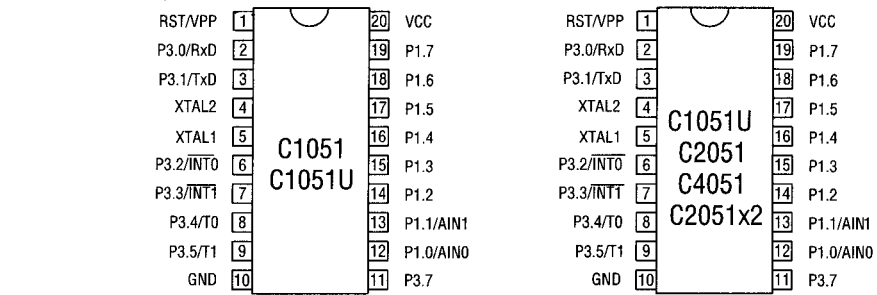
Przegląd rysunków przedstawiających rozmieszczenie wyprowadzeń wybranych mikrokontrolerów rodziny '51 rozpoczęto od obudów dwurzędowych – DIP (wyprowadzenia w rastrze 2,54 mm) i SOP (elementy do montażu powierzchniowego o wyprowadzeniach w rastrze z zakresu 0,65...1,27 mm, zależnym od konkretnego układu i producenta – rys. D.1 i D.2). Na rys. D.3 przedstawiono rozmieszczenie wyprowadzeń w mikrokontrolerach z obudowami LCC (PLCC lub CLCC – raster wyprowadzeń 1,27 mm) oraz QFP (elementy do montażu powierzchniowego o wyprowadzeniach w rastrze 0,8 lub 1,0 mm).



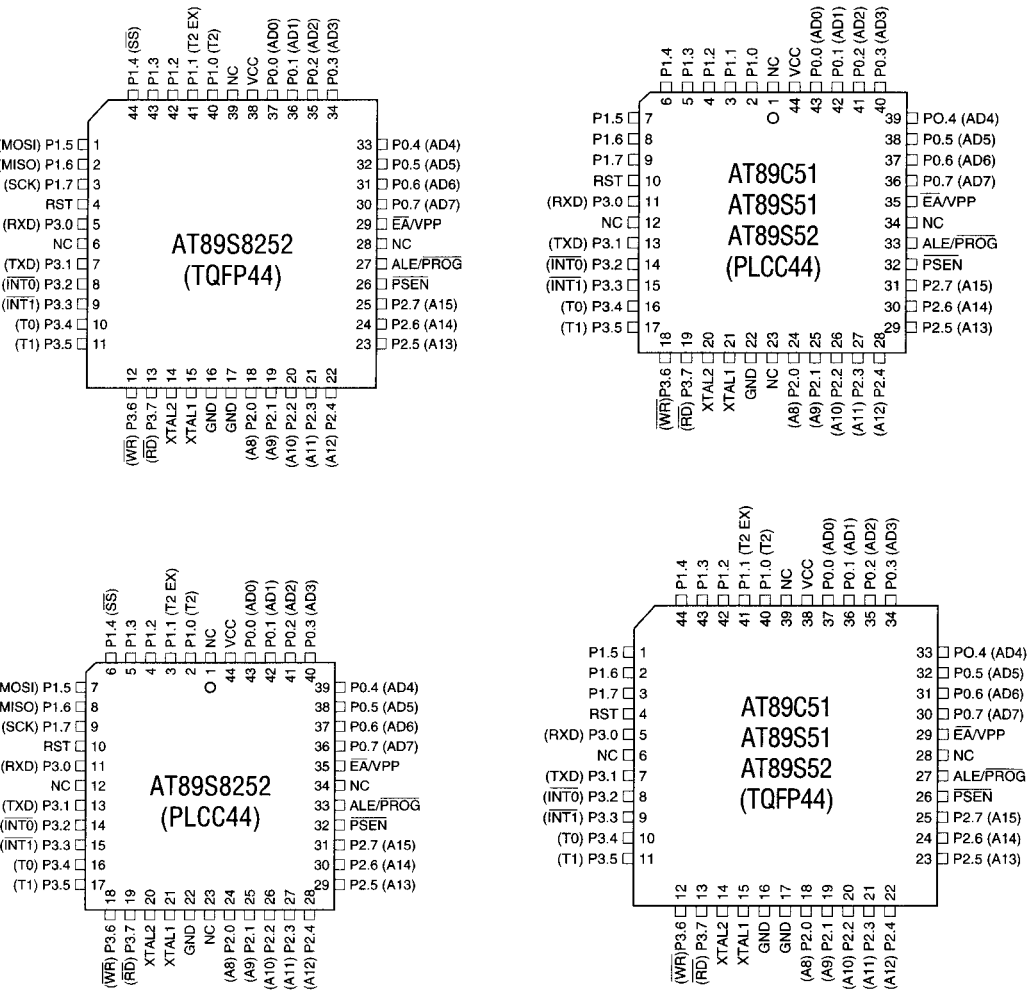
Mikrokontrolery w obudowach LCC produkowane niegdyś przez firmę AMD wykorzystują wyprowadzenie o numerze 1 jako końcówkę dodatniego napięcia zasilania – dotyczy to m.in. układów C51 produkowanych przez tę firmę. Rozwiązanie to stanowi istotne odstępstwo od reguł stosowanych przez innych producentów, którzy pozostawiają omawiane wyprowadzenie niepodłączone (NC) lub połączone z masą układu (GND).



Rys. D.1. Rozmieszczenie wyprowadzeń wybranych mikrokontrolerów w obudowach DIL-40



Rys. D.2. Rozmieszczenie wyprowadzeń wybranych mikrokontrolerów w obudowach DIL-20



Rys. D.3. Rozmieszczenie wyprowadzeń wybranych mikrokontrolerów w obudowach TQFP/PLCC-44

Dodatek E. Tablice kodów ASCII

DEC	HEX	CHAR
0	00	NUL
1	01	SOH
2	02	STX
3	03	EXT
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	SP
33	21	!
34	22	"
35	23	#
36	24	\$
37	25	%
38	26	&
39	27	`
40	28	(
41	29)

DEC	HEX	CHAR
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	Ø
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S

DEC	HEX	CHAR
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
100	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	
125	7D	}

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
126	7E	~	172	AC	¼	212	D4	⌚
127	7F	DEL	173	AD	½	213	D5	⌚
128	80	Ç	174	AE	«	214	D6	⌚
129	81	ù	175	AF	»	215	D7	⌚
130	82	é	176	B0	☐	216	D8	⌚
131	83	â	177	B1	■	217	D9	⌚
132	84	ä	178	B2	■	218	DA	⌚
133	85	à	179	B3	⋮	219	DB	■
134	86	å	180	B4	⌚	220	DC	■
135	87	ç	181	B5	⌚	221	DD	■
136	88	ê	182	B6	⌚	222	DE	■
137	89	ë	183	B7	⌚	223	DF	■
138	8A	è	184	B8	⌚	224	E0	α
139	8B	í	185	B9	⌚	225	E1	β
140	8C	î	186	BA	⌚	226	E2	Γ
141	8D	ï	187	BB	⌚	227	E3	π
142	8E	Ä	188	BC	⌚	228	E4	Σ
143	8F	Å	189	BD	⌚	229	E5	σ
144	90	É	190	BE	⌚	230	E6	∞
145	91	æ	191	BF	⌚	231	E7	τ
146	92	Æ	192	C0	⌚	232	E8	Φ
147	93	ô	193	C1	⌚	233	E9	θ
148	94	í	194	C2	⌚	234	EA	Ω
149	95	ò	195	C3	⌚	235	EB	δ
150	96	û	196	C4	-	236	EC	∞
151	97	ü	197	C5	†	237	ED	Ø
152	98	ÿ	198	C6	‡	238	EE	€
153	99	Ö	199	C7	‡	239	EF	∩
154	9A	Ü	200	C8	⌚	240	F0	≡
155	9B	ø	201	C9	⌚	241	F1	±
156	9C	£	202	CA	⌚	242	F2	≥
157	9D	¥	203	CB	⌚	243	F3	≤
158	9E	Pt	204	CC	⌚	244	F4	¡
159	9F	ƒ	205	CD	=	245	F5	
160	A0	á	206	CE	≠	246	F6	+
161	A1	í	207	CF	⌚	247	F7	≈
162	A2	ó	208	D0	⌚	248	F8	°
163	A3	ú	209	D1	⌚	249	F9	•
164	A4	ñ	210	D2	⌚	250	FA	•
165	A5	Ñ	211	D3	⌚	251	FB	√
166	A6	æ				252	FC	η
167	A7	º				253	FD	2
168	A8	¿				254	FE	•
169	A9	⌚				255	FF	
170	AA	¬						
171	AB	½						

E.2. Znaki i odpowiadające im kody rozszerzonego standardu ASCII (ich zawartość zależy od używanej strony kodowej/standardu kodowania)

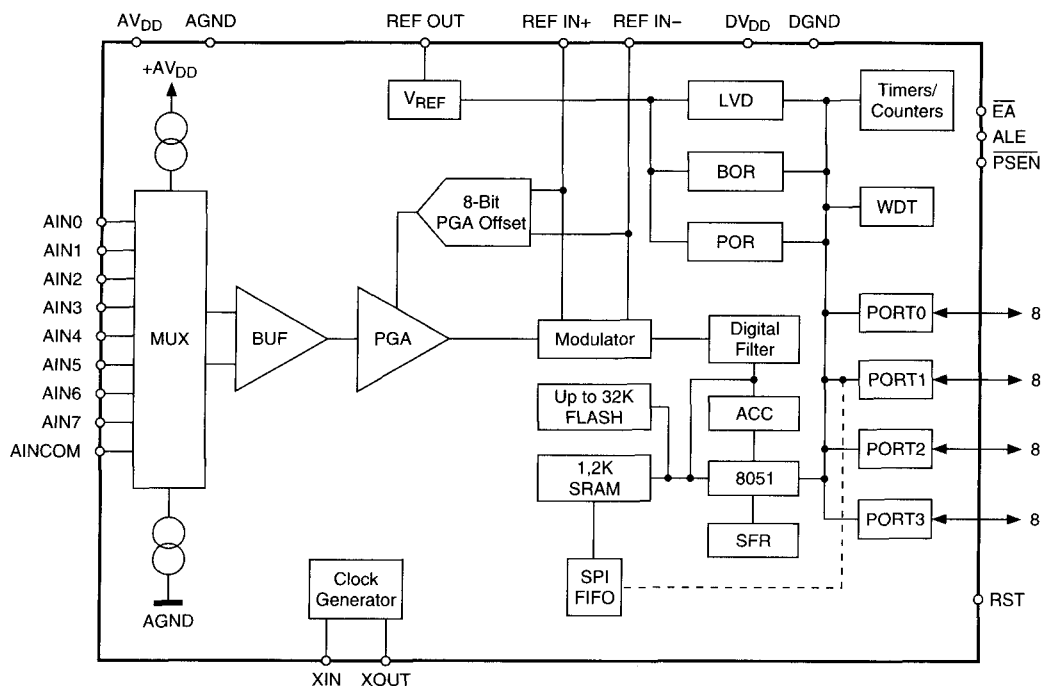
4 starsze bity młodsze bity	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	A	P	`	F				-	9	3	o	p
xxxx0001	(2)		!	1	A	Q	a	4			.	7	7	4	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	×	ρ	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	ω
xxxx0100	(5)		\$	4	D	T	d	t			、	エ	ト	ホ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	℃	Ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ア	キ	ヌ	ラ	g	π
xxxx1000	(1)		(8	H	X	h	x			ィ	ク	ネ	リ	フ	×
xxxx1001	(2))	9	I	Y	i	y			ゝ	ケ	ル	ル	”	γ
xxxx1010	(3)		*	:	J	Z	j	z			エ	コ	ン	レ	j	〒
xxxx1011	(4)		+	;	K	L	k	l			オ	サ	ヒ	ロ	*	⌘
xxxx1100	(5)		,	<	L	¥	l	l			カ	シ	フ	ワ	Φ	⌘
xxxx1101	(6)		-	=	M	J	m	}			ユ	ズ	ヘ	ン	⌘	÷
xxxx1110	(7)		.	>	N	^	n	+			ヨ	セ	ホ	”	⌘	
xxxx1111	(8)		/	?	O	_	o	+			ッ	リ	マ	”	ö	■

**E.3. Kody znaków zapisanych w generatorze znaków sterownika LCD HD44870 (wersja standardowa).
Znaki o kodach 00...0Fh (CG RAM1.....CG RAM8) użytkownik może samodzielnie zdefiniować**

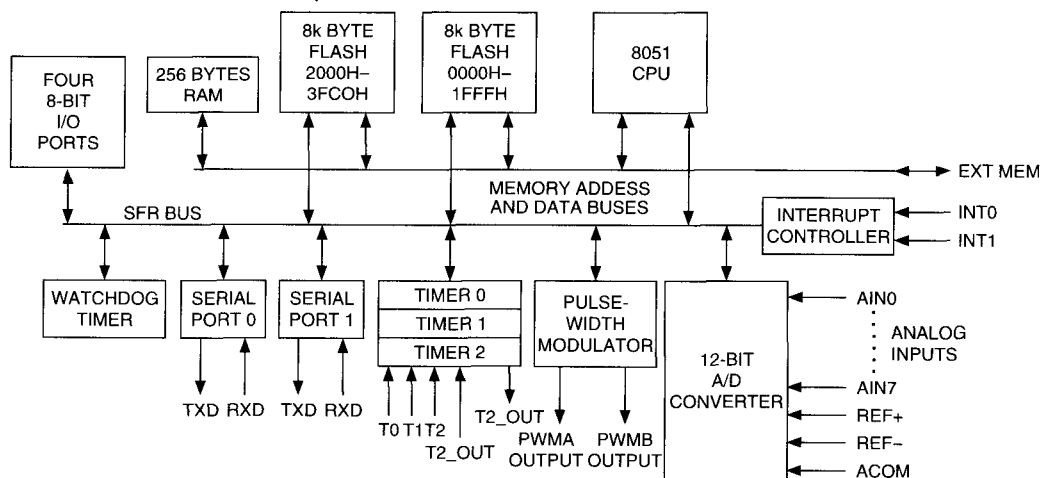
4 starsze bity młodsze bity		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)	␣		0	1	A	B	C	D	E	F	G	H	I	J	K	L
xxxx0001	(2)	!	"	3	4	5	6	7	8	9	:	;	<	=	?	@	A
xxxx0010	(3)	"	#	2	B	R	b	r	W	Γ	φ	2	À	Ó	ä	ö	
xxxx0011	(4)	"	#	3	C	S	c	s	3	π	ε	3	À	Ó	ä	ö	
xxxx0100	(5)	␣	\$	4	D	T	d	t	H	Σ	×	ℙ	À	Ó	ä	ö	
xxxx0101	(6)	3	%	5	E	U	e	u	Ÿ	σ	¥	ℙ	À	Ó	ä	ö	
xxxx0110	(7)	␣	&	6	F	V	f	v	Π	Δ	!	9	Æ	Ö	ø	ö	
xxxx0111	(8)	4	'	7	G	W	w	Π	τ	3	.	Ç	×	ç	÷		
xxxx1000	(1)	↑	(8	H	X	h	x	Y	*	†	ω	É	⊗	é	⊗	
xxxx1001	(2)	↓)	9	I	Y	i	y	U	Θ	1		É	Ò	é	Ò	
xxxx1010	(3)	÷	*	:	J	Z	j	z	4	Ω	Ω	Ω	É	Ó	é	Ó	
xxxx1011	(4)	←	+	:	K	Γ	k	(W	Δ	×	×	É	Ò	é	Ò	
xxxx1100	(5)	<	,	<	L	\	l	l	W	∞	W	W	i	Ü	i	Ü	
xxxx1101	(6)	>	-	=	M	I	m	>	b	⊗	9	W	i	Ý	i	Ý	
xxxx1110	(7)	␣	.	>	N	^	n	~	W	ε	Q	W	i	ß	i	ß	
xxxx1111	(8)	␣	/	?	O	_	o	Δ	3	Π	°	¿	i	ß	i	ß	

E.4. Kody znaków zapisanych w generatorze znaków sterownika LCD HD44870 (wersja europejska).
Znaki o kodach 00...0Fh (CG RAM1.....CG RAM8) użytkownik może samodzielnie zdefiniować

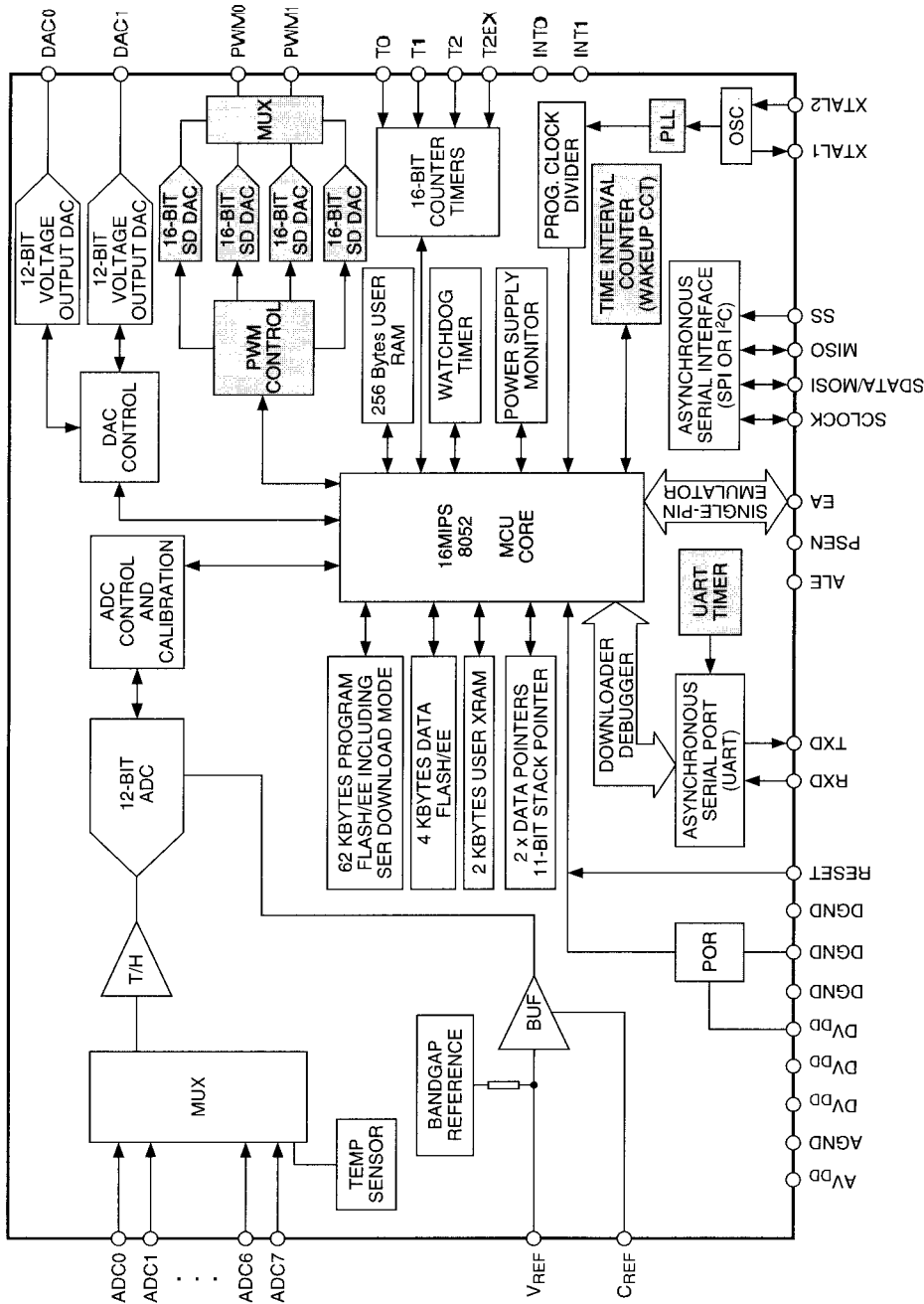
Dodatek F. Schematy blokowe wybranych mikrokontrolerów rodziny '51



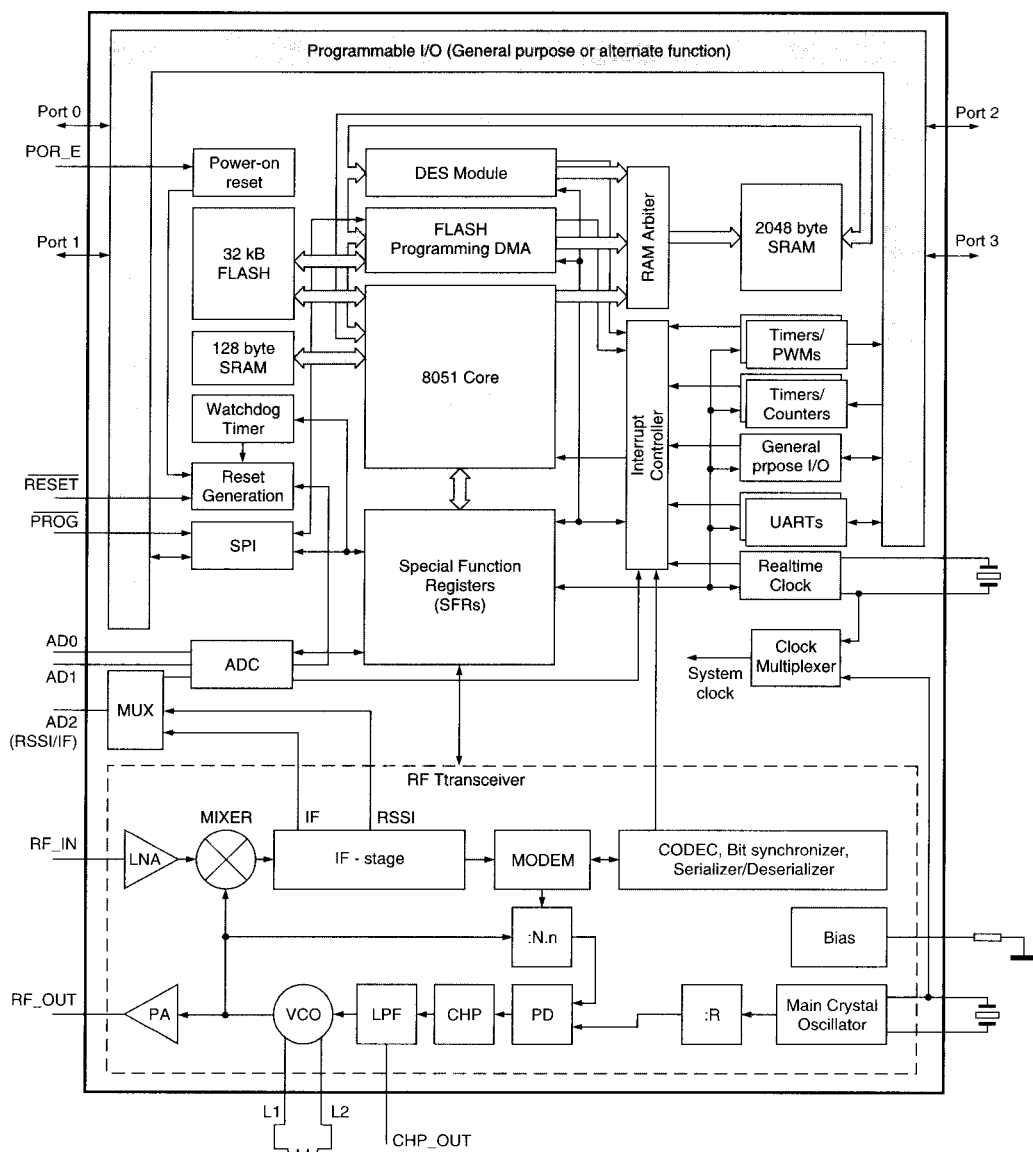
Rys. F.1. Schemat blokowy mikrokontrolera MSC1210 produkowanego przez firmę Texas Instruments (Burr-Brown)



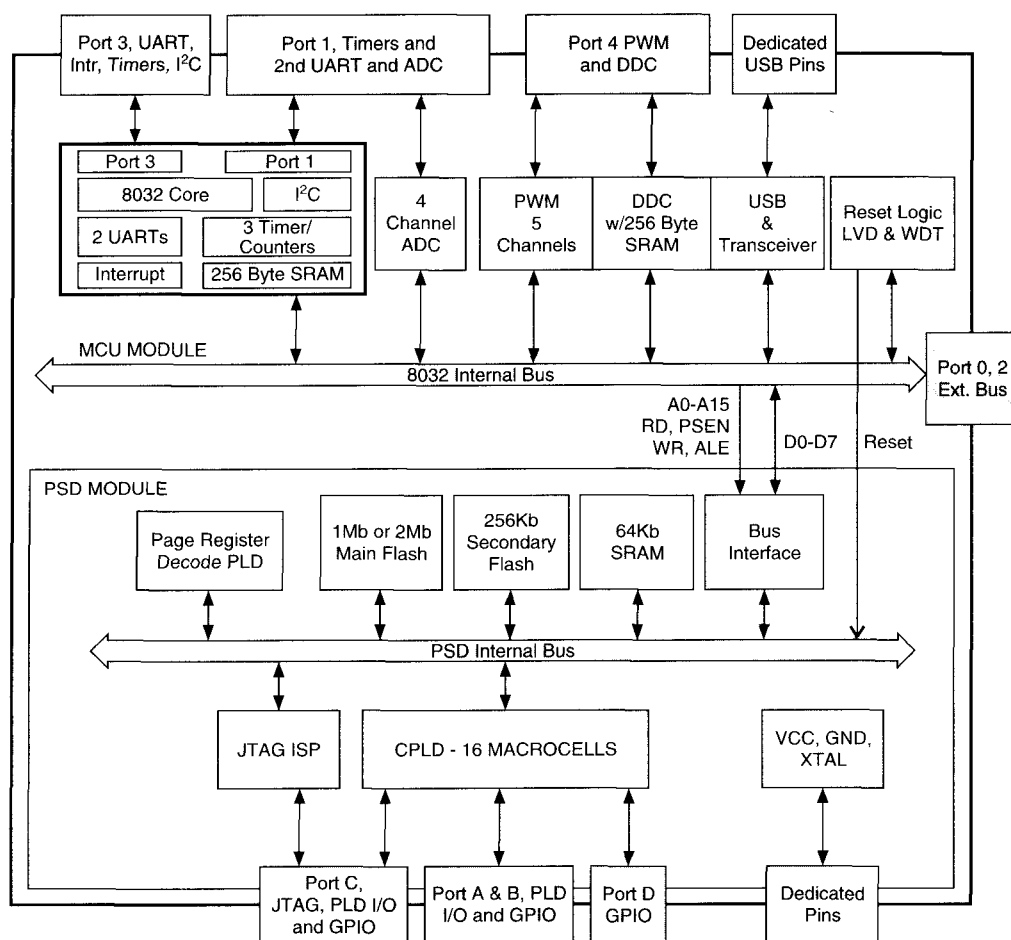
Rys. F.2. Schemat blokowy mikrokontrolerów MAX7651/7652 produkowanych przez firmę Maxim



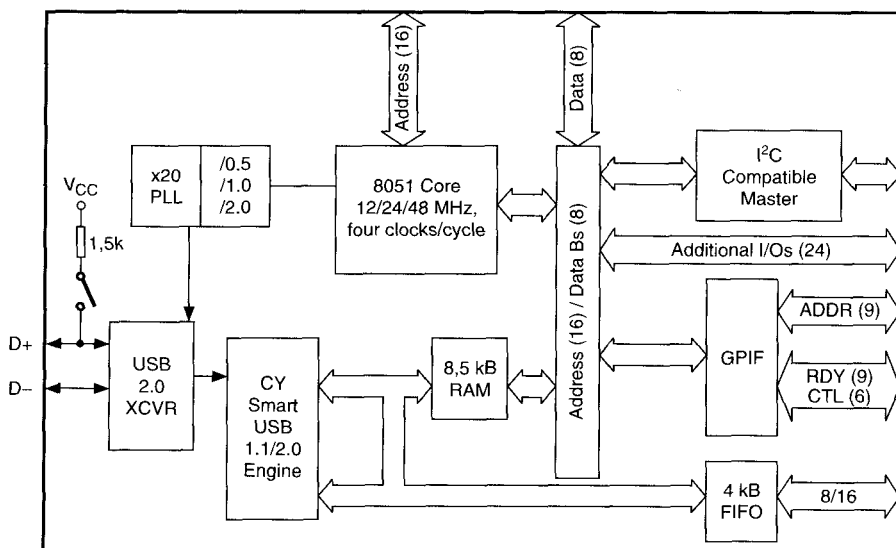
Rys. F.3. Schemat blokowy mikrokontrolera AduC824 produkowanego przez firmę Analog Devices



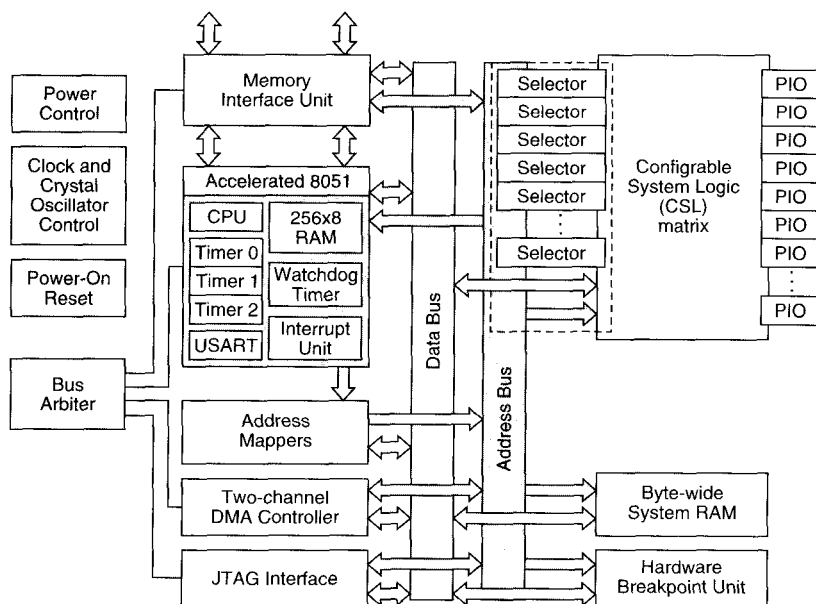
Rys. F.4. Schemat blokowy mikrokontrolera CC1010 produkowanego przez firmę Chipcon



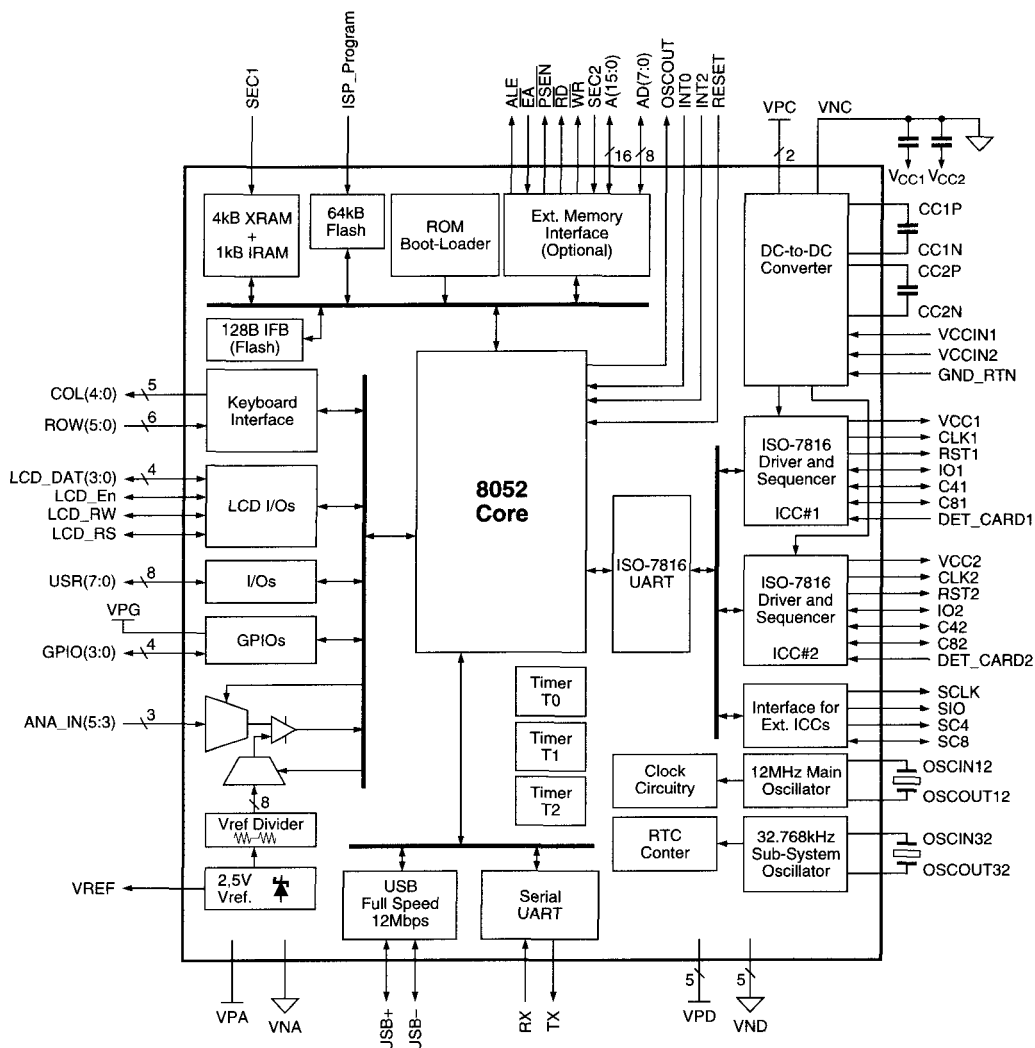
Rys. F.5. Schemat blokowy mikrokontrolera μ PSD3200 produkowanego przez firmę ST Microelectronics



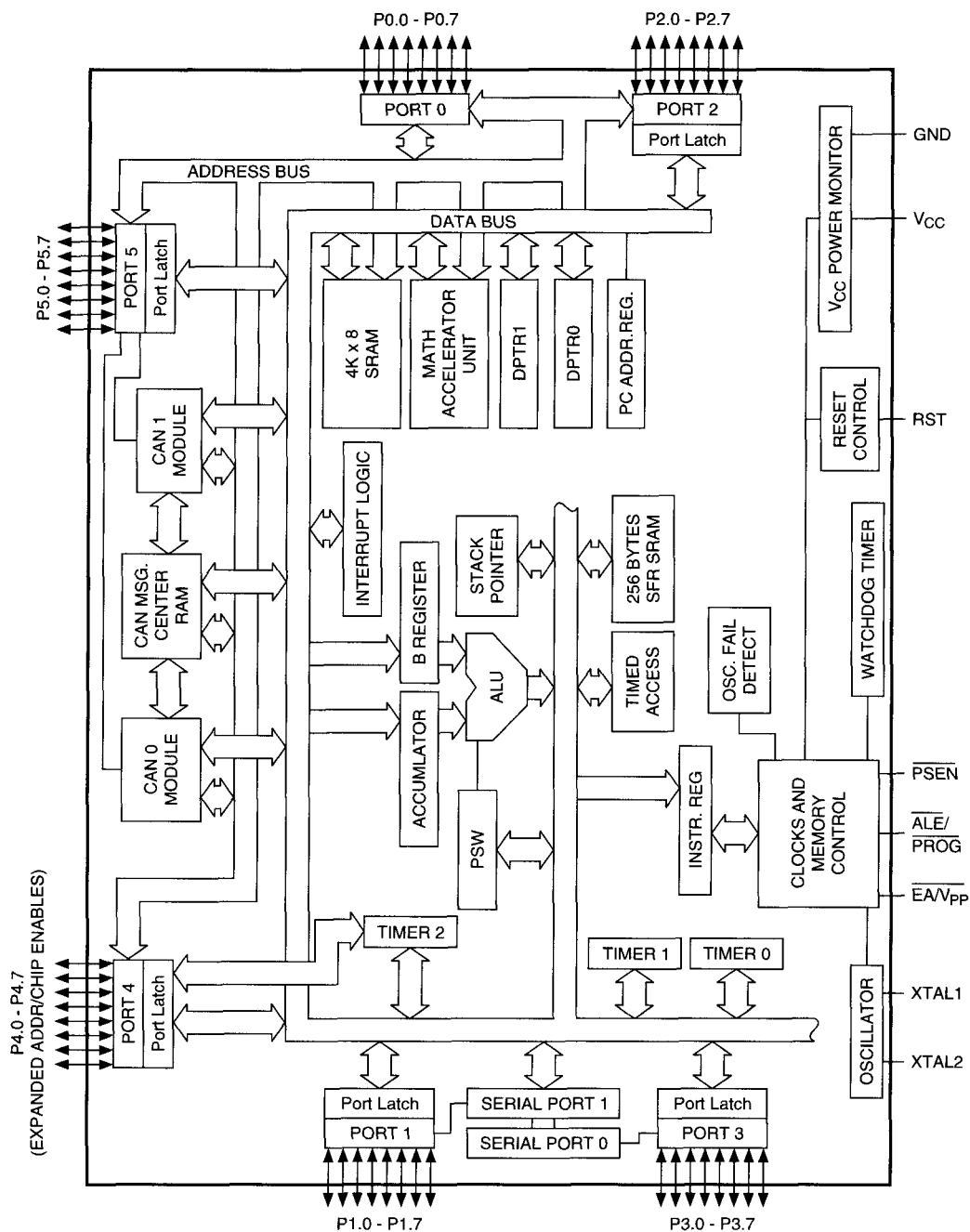
Rys. F.6. Schemat blokowy mikrokontrolera CY7C68013 produkowanego przez firmę Cypress



Rys. F.7. Schemat blokowy mikrokontrolerów z rodziny E5 firmy Triscend



Rys. F.8. Schemat blokowy mikrokontrolera 73S1121F produkowanego przez firmę TDK



Rys. F.9. Schemat blokowy mikrokontrolera DS80C390 produkowanego przez firmę Maxim (Dallas)

- 1-wire 239, 250
- 80C31 195, 263
- 80C32 263
- 80C51 263
- 80C52 264
- 80C58 158, 268
- 87C51 76, 263
- 87C52 158, 264
- 87C54 158, 267
- 89C1051 78, 268
- 89C1051U 269
- 89C2051 191, 269, 290
- 89C4051 78, 191, 270
- 89C51 78, 81, 263
- 89C52 78, 191, 264
- 89C54 267
- 89C55 267
- 89LS53 192, 266
- 89LV51 78, 83, 263
- 89LV52 76, 81, 265
- 89S51 82, 264
- 89S53 82, 188
- 89S8252 82, 249, 270
- ACALL 92, 128
- ACC 16, 90, 157
- ACK 225
- ADD 93
- ADDC 94
- Adresowanie 10-bitowe 231
 - 16-bitowe 122
 - 7-bitowe 118
 - 8-bitowe 122
 - bezpośrednie 16, 90
 - natychmiastowe 90
 - pośrednie 90
 - pośrednie zawartością rejestru 90
- AJMP 95
- ALE 27, 33, 67
- ALU 9
- ANL 96
- Arbitraż 223, 226
- ASCII 205, 280
- Asembler 176
- AUXR 16, 157
- Banki rejestrów 13, 90, 163
- Bascom 178
- Bit startu 44, 49, 215
- Bit stopu 44, 49, 165, 215
- Blokada przerwań 59, 61
- Budowa portów i/o 23
- Busy flag 210, 212
- CCU 34
- CG RAM 209
- CJNE 99
- CLR 101
- CMOS 17, 215, 224
- CPL 102
- CPU 9, 144
- Cursor shift 209
- Cykl maszynowy 12, 18
- DA 103
- DD RAM 208
- Debugger 176, 184
- DEC 104
- Display clear 208
 - on/off 209
- DIV 106
- DJNZ 106
- DPTR 15, 22, 28, 54
- Drgania styków 199
- EA 12, 75, 160
- EEPROM 10, 53
- Emulator 183
- Entry mode 208
- EPROM 74, 184
- FEEPROM 74
- Flash 74, 78
- Function set 209
- GPR 14
- Grape 180
- HD44870 282
- I²C 222
- Idle 66
- IDLS 67

- INC 108
INT 63
ISP 88, 187, 192
- JB 109
JBC 110
JC 111
JMP 111
JNB 112
JNC 113
JNZ 113
JZ 114
- Keil 177
Komparator analogowy 53
- LCALL 115
Licznik 36, 39, 41
Liczniki czuwające 55
Kista rozkazów 89
LJMP 116
- Łącze szeregowo 43, 49
- Master 239
MISO 50, 166
Monitor 184
MOSI 50, 166
MOV 116
MOVC 121
MOVX 28, 54, 122
MUL 123
- Napięcie programowania 76, 82
– zasilania 22, 68, 72, 158, 163
NMOS 17, 32, 66
NOP 124
- Obciążalność portów 32
Obsługa klawiatury 196
– przerwań 61
– wyświetlacza LCD 179, 205
– wyświetlaczy LED 201
Organizacja pamięci 12
ORL 124
OTP 74
- Pamięć danych 13, 83
– programu 12
PCA 34
PCON 47, 56, 67, 162
Pobór mocy 66
POP 127
Porty wejściowo-wyjściowe 10, 23
Power Down 67
Prędkość transmisji 43, 47
Priorytet przerwania 60, 160
Programator 187
– ISP 88, 187
Programatory stacjonarne 188
Przerwania wewnętrzne 56
– zewnętrzne 56, 63, 199
PSEN 28
PSW 15, 91, 163
PUSH 128
- Quick 74
- RAM 9, 13, 20, 67, 90
Ramka danych 48
RET 128
RETI 62, 67, 129
Rezystor podciągający 13, 20, 25, 53, 196,
224, 235, 251
RL 129
RLC 130
RR 130
RRC 131
RS 207
RS232 185, 215, 246
RST 21, 56
RTC 243
RxD 43
- SCK 50, 84, 189
SCL 223, 235
SCON 44, 165
SDA 223, 235
SDCC 180
SETB 131
SFR 13, 16, 144
SJMP 132
Slave 239
Slow Down 66

-
- SP 15, 166
SPCR 52, 166
SPI 50, 84, 219
SPSR 167
START 224
STOP 224
Stos 15
SUBB 133
SWAP 134
Symulatory programowe 183
- T2MOD 41, 172
Taktowanie mikrokontrolera 17
TCON 36, 168
Timer 36
TMOD 36, 171
Tryby adresowania 90
TxD 43
- Układy czuwające 55
USART
Uśpienie 66
- Watchdog 55
WCON 55, 172
Wektor przerwania 58
Wewnętrzna pamięć danych 83, 90
Wired AND 239
WireIt 180
WMCON 16, 54, 173
Wskaźnik danych 15, 28,
– stosu 15
Wyświetlanie multipleksowe 201
- XCH 135
XCHD 136
XRL 136
- Zamrożenie 66
Zerowanie 20, 101
Zewnętrzna pamięć danych 122, 184, 195
Zgłoszenie przerwania 9, 59, 62
- Źródło przerwania 47, 58