Zdzisław Rochala

Architektura mikrokontrolerów i mikrokomputerów urządzeń mechatronicznych

Warszawa 2011

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

fundusze unijne dla oświaty

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Politechnika Warszawska
Wydział Samochodów i Maszyn Roboczych
Studia Podyplomowe dla Nauczycieli Przedmiotów Zawodowych
02-524 Warszawa, ul. Narbutta 84, tel. 22 849 43 07, 22 234 83 48
ipbmvr.simr.pw.edu.pl/spin/, e-mail: sto@simr.pw.edu.pl

Opiniodawca: mgr inż. Zdzisław DĄBROWSKI

Projekt okładki: Norbert SKUMIAŁ, Stefan TOMASZEK

Projekt układu graficznego tekstu: Grzegorz LINKIEWICZ

Skład tekstu: Janusz BONAROWSKI

Publikacja bezpłatna, przeznaczona dla słuchaczy Studiów Podyplomowych dla Nauczycieli Przedmiotów Zawodowych.

ISBN 83-89703-77-7

Druk i oprawa: Drukarnia Expol P. Rybiński, J. Dąbek Spółka Jawna,
87-800 Włocławek, ul. Brzeska 4

Spis treści

Wstęp

Niniejsze materiały zostały opracowane w ramach realizacji Programu Rozwojowego Politechniki Warszawskiej współfinansowanego przez Unię Europejską w ramach Europejskiego Funduszu Społecznego - PROGRAM OPERACYJNY KAPITAŁ LUDZKI. Przeznaczone są dla słuchaczy Studiów Podyplomowych „MECHATRONIKA POJAZDÓW I MASZYN" prowadzonych na Wydziale Samochodów i Maszyn Roboczych Politechniki Warszawskiej.

Niniejsze opracowanie przygotowano dla przedmiotu pt. „Architektura mikrokontrolerów i mikrokomputerów urządzeń mechatronicznych". Jego zawartość merytoryczna w pełni odpowiada zakresowi opisanemu w sylabusie opracowanym dla tego przedmiotu.

Całość opracowanych materiałów dydaktycznych dla ww przedmiotu zawarta została w 5 rozdziałach. Rozdział 1 został poświęcony na określenie obszarów zastosowań mikroelektroniki w systemach mechatronicznych. W rozdziale 2 zdefiniowano podstawowe pojęcia związane z techniką komputerową oraz omówiono budowę i zasadę działania mikroprocesora. Rozdział 3 poświęcony jest organizacji systemu mikroprocesorowego i charakterystyce jego poszczególnych elementów składowych. W rozdziale 4 przedstawiono pojęcia związane z programowaniem systemów mikroprocesorowych. Przedstawiono krótką charakterystykę języka programowania Bascom oraz narzędzia wspomagające proces tworzenia oprogramowania i uruchomiania systemu mikroprocesorowego. W ostatnim rozdziale 5 przedstawiono i omówiono przykłady programów obsługi najczęściej wykorzystywanych przez użytkowników wbudowanych układów peryferyjnych i standardowych urządzeń zewnętrznych systemu mikroprocesorowego.

Zajęcia dydaktyczne zdecydowanej większości przedmiotów składających się na program studiów będą realizowane, oprócz wykładu, także w formie ćwiczeń laboratoryjnych prac projektowych. Dlatego istotną częścią tych materiałów, oprócz prezentacji materiału teoretycznego, są opisy przebiegu ćwiczeń wykonywanych podczas zajęć dydaktycznych oraz propozycje zadań do samodzielnego wykonania przez słuchaczy. Tak skonstruowane materiały dydaktyczne pomogą słuchaczom w nabyciu praktycznych umiejętności z zakresu posługiwania się technikami komputerowymi niezbędnych w realizacji współczesnych procesów projektowo wytwórczych.

Dopuszcza się również możliwość przygotowania oddzielnego opracowania w formie przewodnika do ćwiczeń projektowych. Informacje na ten temat powinny być zamieszczone we „Wstępie" do podstawowych materiałów opracowanych dla przedmiotu. Ponadto należy wspomnieć o tym, że materiały uzupełniające i aktualizujące do przedmiotu będą udostępniane studentom za pośrednictwem systemu e-learning.

1 Obszary zastosowań techniki mikroprocesorowej (mikroelektroniki) w systemach mechatronicznych maszyn i pojazdów

W tym rozdziale

1.1. Elektronika w motoryzacji

Elektronika [7, 13] zrewolucjonizowała technikę samochodową. Początkowo elementy mechaniczne zastępowano elektronicznymi, by poszczególne układy uczynić bardziej niezawodnymi, na przykład usunięto styki przerywacza w układzie zapłonowym. Stopniowo w samochodach przybywało coraz więcej nowych układów, które bez zastosowania elektroniki w ogóle nie byłyby do pomyślenia.

Historia elektroniki samochodowej rozpoczęła się w 1958 roku wraz z zaprojektowaniem wielopunktowego wtrysku paliwa MPI (ang. *Multi Point Injection*) dla silników z zapłonem iskrowym. System ten, który opracowały wspólnie amerykańskie firmy Chrysler i Bendix, sterował odpowiednim wstrzykiwaniem paliwa przez osobne dla każdego cylindra wtryskiwacze, umieszczone w kolektorze, przed zaworem dolotowym. Wcześniej wszystkie systemy samochodów, w tym systemy wtrysku paliwa, kontrolowane były wyłącznie w sposób mechaniczny bądź elektryczny.

Prekursorem w zastosowaniach elektroniki w motoryzacji jest niemiecka firma Bosch. W 1979 roku jej urządzenie o nazwie Motronic zostało zamontowane w dwóch modelach samochodów produkowanych przez znaną firmę motoryzacyjną BMW. Pełniło ono rolę urządzenia nadzorującego i sterującego prawidłowym doborem mieszanki paliwowej. Zajmowało się również sterowaniem momentu wyzwolenia iskry na świecach zapłonowych. Sterownik z 1979 roku ważył ponad kilogram, na cały system składało się ponad 290 różnych elementów tworzących odpowiednik dzisiejszego systemu mikroprocesorowego. Do roku 1996 postęp, jaki dokonał się w elektronice, zaznaczył się spadkiem masy Motronica do 250 gramów oraz znaczną redukcją liczby zastosowanych w nim części – obecnie prawidłowe działanie gwarantuje jedynie osiemdziesiąt elementów. Duża w tym zasługa mikroprocesorów jednoukładowych i montażu powierzchniowego.

Jednak na prawdziwą cyfrowo-elektroniczną rewolucję przyszło poczekać przeszło 20 lat. Otóż dopiero w 1979 roku firma Bosch opracowała mikroprocesorowy system sterowania samochodów o nazwie Motronic. Znalazł on zastosowanie w samochodach BMW 732 i BMW 633 Csi. Do zadań tego 4-bitowego komputera, składającego się z 290 elementów i ważącego 1,14 kg należało wyłącznie sterowanie układami wtrysku

i zapłonu. Z dzisiejszej perspektywy można powiedzieć, że był to pierwszy na świecie system ECU (ang. *Engine Control Unit*), czyli komputer sterujący i nadzorujący pracę silnika.

Mimo coraz mniejszych wymiarów, ilość czynności, za które odpowiadają w samochodzie mikrosterowniki, gwałtownie rosła.

Układy mikroprocesorowe już od dawna kontrolują pełny wtrysk paliwa do silnika w połączeniu z kontrolą czystości spalin. Zajmują się też sterowaniem systemami przeciwpoślizgowymi ABS, pełnią funkcję kontroli odstępu między pojazdami czy nawet ustalaniem parametrów stopnia amortyzacji pojazdu. Układy komputerowe zajmują się również (a może – przede wszystkim) uprzyjemnianiem życia kierowców. Nadzorują pracę klimatyzacji, sterują otwieraniem i zamykaniem okna dachowego, włączają w odpowiednim momencie wycieraczki. Pełnią funkcję komputerów pokładowych informujących kierowcę o tak ważnych rzeczach jak temperatura na zewnątrz pojazdu, ilość paliwa w zbiorniku, jego średnie zużycie itd., jak również funkcję zabezpieczenia przeciwkradzieżowego. Dzięki odpowiednim mocom obliczeniowym i specjalnym algorytmom, włączenie czy wyłączenie alarmu oraz odblokowanie zapłonu odbywa się z wykorzystaniem zmiennego kodu sterującego. Ze względu na jego złożoność jest on bardzo trudny do "złamania" przez złodzieja. Stosowane są również wszelkiego rodzaju układy identyfikacji bezstykowej. Ich elementy zamontowane są w fabrycznych kluczykach pojazdu.

Nowoczesne samochody, zwłaszcza te należące do wyższego, luksusowego segmentu rynku wręcz naszpikowane są elektroniką. Do jej zadań należy nie tylko kontrola elementów związanych z śledzeniem kąta wyprzedzenia zapłonu czy monitorowanie temperatury i ciśnienia oleju w silniku, ale sterują one szeregiem interaktywnych podzespołów ułatwiającą kierowcy prowadzenie auta takich jak np. ABS (ang. *Anti-Lock Braking System*) czy ASR (ang. *Acceleration Slip Regulation*). Komputer pokładowy informuje również kierowcę o zużyciu paliwa, ilości pozostałej benzyny, przejechanej trasie czy o takich parametrach jak o braku zapięcia pasów, niezapaleniu świateł, a nawet, co coraz częściej się zdarza, o prawidłowym ciśnieniu w oponach.

Komputeryzacja samochodów i systemy kontroli trakcji oraz mikroprocesorowe mechanizmy zapewniające bezpieczeństwo czynne oraz bierne coraz częściej trafiają nie tylko do luksusowych limuzyn, ale spotyka się je już w sporej części tańszych i średniej klasy pojazdów. Praktycznie standardem stało się stosowanie wspomnianych przed chwilą systemów ABS i ASR, elektronicznie sterowanych poduszek powietrznych oraz

mniej lub bardziej zaawansowanych komputerów pokładowych. Warto więc przyjrzeć się bliżej, jakie komputerowe systemy znaleźć można we współczesnych samochodach, kiedy zaczęto je stosować, jak się one między sobą komunikują oraz co nas czeka w najbliższej przyszłości.

Oprócz wymienionych układów elektronicznego sterowania funkcjami silnika oraz systemów ABS i ASR we współczesnych samochodach, w zależności od ich marki, ceny, modelu i klasy, montowane są również układy sterujące innymi jego podzespołami. Są to zwykle sterowniki skrzyni biegów montowane w modelach wyposażonych w automatyczną lub półautomatyczną skrzynię, układy sterowania światłami, układy zarządzania klimatyzacją, elementy sterowania siedzeniami oraz sterownik otwierania drzwi i elektrycznych szyb, moduł odpowiedzialny za sterowanie poduszkami powietrznymi i napinaniem pasów, zespół sterowania instalacją alarmową, moduł obsługi telefonu, detektor martwego pola itp.

Mimo różnic konstrukcyjnych, ogólny mechanizm działania tych podzespołów jest bardzo podobny. W większości wypadków składają się one z trzech elementów: sterownik, czujniki i elementy wykonawcze. Czujniki i elementy wykonawcze różnią się w zależności od funkcji, którą wykonują. Mikroprocesory sterujące są natomiast do siebie bardzo podobne - często jest to ten sam układ, który został odpowiednio zaprogramowany do wykonywania określonych czynności. Wyjątkiem są procesory wykorzystywane do budowy centralnego komputera, które wyposażone są zwykle w sterownik ekranu LCD.

Układy mikroprocesorowe stosowane do budowy poszczególnych funkcjonalnych modułów to przeważnie 8-bitowe nieskomplikowane procesory. Przykładem takiego układu jest uniwersalna rodzina kości S08 produkowanych przez Freescale Semiconductor. Są to 40-megahercowe (z 20 MHz magistralą systemową) układy korzystające z zestawu instrukcji HC08. Zintegrowano w nich 60 KB pamięć flash, 2 KB EEPROM i 4 KB pamięć RAM. Podobną konstrukcję ma nowa rodzina układów firmy STMicroelectronics - STM8A. Są to 16-megahercowe układy wyposażone w pamięć flash (od 8 do 256 KB, w zależności od wersji), od 384 B do 4 KB pamięć EEPROM i od 1 do 12 KB pamięci RAM. Układy te mają też szeroką tolerancję napięć wejściowych - mogą być zasilane bowiem napięciem od 3,0 do 5,5 V.

Bardziej zaawansowane i szybsze układu 16- i 32-bitowe wykorzystuje się przede wszystkim, oprócz wspomnianych przed chwilą systemów ECU, do budowy systemów ABS i ASR oraz innych kluczowych z punktu widzenia kontroli trakcji systemów, w których decyzje muszą być podejmowane na bieżąco. Warto w tym miejscu zaznaczyć, że

układy przeznaczone dla przemysłu samochodowego muszą charakteryzować się dużą odpornością na działanie czynników zewnętrznych. Najważniejsza jest duża odporność na niskie i wysokie temperatury. Układ musi być w stanie pracować w temperaturach od -40 do 150°C.

1.2. Systemy mechatroniczne maszyn i pojazdów

Nowoczesne maszyny i urządzenia możemy traktować jako systemy mechatroniczne [17]. Dobrym przykładem systemu mechatronicznego jest nowoczesny samochód. Współczesne samochody osobowe i użytkowe są wypełnione układami mechatronicznymi (układy mechaniczne sterowane systemami elektronicznymi), które obejmują następujące zakresy zastosowań:

- silnik z układem napędowym;
- bezpieczeństwo;
- komfort;
- komunikacja i multimedia.

Układy elektroniczne służące do sterowania układów mechanicznych dzieli się na następujące bloki funkcjonalne:

· czujniki i nadajniki wartości zadanej;

· sterownik (sterowniki);

· elementy wykonawcze (nastawniki);

· komunikacja między sterownikami (sieć magistrali danych);

· elektroniczna diagnostyka.

Czujniki rejestrują warunki pracy (np. prędkość obrotową silnika, prędkość obrotową kół, temperaturę). Przekształcają one wielkości fizyczne na sygnały elektryczne. Nadajniki wartości zadanej (np. włączniki uruchamiane przez kierowcę) zadają określone nastawy.

Sterownik przetwarza informacje uzyskiwane z czujników i nadajników wartości zadanej według określonych matematycznych metod obliczeń (algorytmów sterowania i regulacji). Steruje on elementy wykonawcze elektrycznymi sygnałami wyjściowymi. Ponadto sterownik jest interfejsem dla innych układów i diagnostyki pojazdu.

Elementy wykonawcze przekształcają elektryczne sygnały wyjściowe sterownika na wielkości mechaniczne. Przykładami elementów wykonawczych są:

- wtryskiwacze silnika benzynowego;
- wtryskiwacze silnika wysokoprężnego;
- silniki elektryczne (np. silnik napędowy podnośnika szyby, silnik napędowy przepustnicy w układzie elektronicznego pedału przyspieszenia EGAS);
- wentylator.

Wobec coraz większej liczby układów elektronicznych zwiększa się również ilość okablowania w samochodzie. Długość wszystkich przewodów współczesnego pojazdu średniej klasy wynosi ok. 1,6 km, a do tego dochodzi 300 złączy wtykowych z około 2000 styków. Połączenie poszczególnych układów w sieć wydatnie zmniejsza całkowitą długość przewodów. Na wspólnej magistrali tylko o dwóch przewodach są przesyłane dane, odczytywane przez wszystkie dołączone do niej odbiorniki. Zaletą sieci jest analizowanie sygnałów z czujników tylko przez jeden sterownik. Na przykład na podstawie wskaźników z różnych wartości prędkości obrotowej kół przesyłanych przez sterownik ABS może być obliczana prędkość pojazdu. To skomplikowane obliczenie uwzględnia także jazdę na zakręcie i poślizg kół. Wartość prędkości jest udostępniana wszystkim odbiornikom przyłączonym do magistrali (np. sterownikowi układu przeciwblokującego, który podejmuje decyzje o hamowaniu poszczególnych kół, sterownikowi silnika, który reguluje prędkość jazdy, radioodbiornikowi samochodowemu, który dostosowuje głośność do prędkości jazdy).

Układ diagnostyki elektronicznej w sterowniku podczas pracy ciągle kontroluje system i poszczególne podzespoły. Informacje o wykrytych usterkach (na przykład zwarcia przewodów, uszkodzenie czujników) są gromadzone w pamięci sterownika. Usterki te mogą być następnie odczytane w stacji obsługi przy użyciu specjalnego testera, podłączonego do złącza diagnostycznego sterownika. Za pomocą testera diagnostycz-

nego można także odczytywać sygnały wysyłane przez czujniki i sterować poszczególne elementy wykonawcze. W ten sposób diagnostyka elektroniczna między innymi ułatwia i przyspiesza wykrywanie usterek w stacji obsługi.

2 Układy wykonawcze w urządzeniach i systemach mechatronicznych

W tym rozdziale

- Komputer, mikrokomputer, mikroprocesor, mikrokontroler
- Budowa, cykl rozkazowy, podstawowe operacje arytmetyczno-logiczne, lista rozkazów typowego mikroprocesora

2.1. Komputer, mikrokomputer, mikroprocesor, mikrokontroler

Trudno dziś wymienić dziedzinę życia, w której nie stosuje się komputerów. Od pierwszych "maszyn matematycznych" ich konstrukcja zmieniła się całkowicie, nadążając za rozwojem techniki, technologii i zapotrzebowania na ich usługi.

Komputer jest urządzeniem elektronicznym przeznaczonym do przetwarzania informacji [39, 47]. Dostarczane dane wejściowe przetwarzane są w nim na określoną postać wynikową. Ze względu na formę wykorzystywanej informacji, komputery dzieli się na:

- komputery **analogowe**;
- komputery **cyfrowe**.

Należy zauważyć, że mimo iż, komputery analogowe znalazły zastosowanie w wielu urządzeniach montowanych w maszynach i pojazdach zostały całkowicie wyparte przez dynamicznie rozwijane na przestrzeni minionych dziesięcioleci komputery cyfrowe.

Główną wadą komputerów analogowych była niewielka dokładność rozwiązania zadania (na poziomie dziesiątych części procenta) wynikająca z dokładności wykonania poszczególnych bloków komputera oraz bardzo mała uniwersalność zastosowań. Komputery te najczęściej wykorzystywano do rozwiązywania równań lub układów równań algebraicznych oraz zwyczajnych równań różniczkowych. Programowanie komputera analogowego polegało na wykonaniu odpowiednich połączeń wejść i wyjść typowych bloków operacyjnych (sumujących, całkujących, różniczkujących, mnożących, dzielących, itd.) użytych do rozwiązania zadania. Cenną, często niedostrzeganą zaletą tych komputerów jest możliwość pracy w rzeczywistej skali czasu, przez co komputer analogowy jest z natury komputerem równoległym. Oznacza to, że wszystkie operacje matematyczne wykonywane są jednocześnie we wszystkich blokach operacyjnych.

Podstawą działania komputerów cyfrowych jest fakt, że matematyczne metody numeryczne pozwalają sprowadzić rozwiązanie dowolnie skomplikowanego zadania do wykonania ciągu operacji arytmetycznych i logicznych wykonywanych nad liczbami. Zatem aby rozwiązać zadanie w komputerze cyfrowym trzeba znaleźć odpowiedni algorytm (sposób rozwiązania zadania), który wyznacza potrzebny ciąg operacji do rozwiązania zadania. Algorytm zapisany w odpowiednim, zrozumiałym dla komputera języku nazywa się **programem**. Ponieważ dla zdecydowanej większości zadań takie algorytmy daje się opracować, to komputery cyfrowe są komputerami uniwersalnymi.

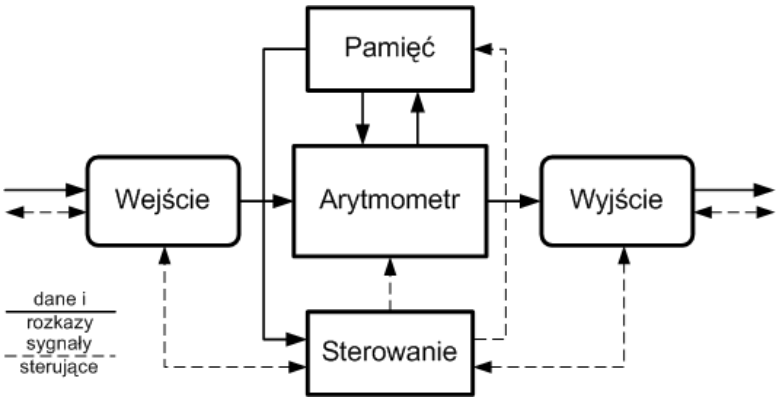
Czas rozwiązania zadania w komputerze cyfrowym zależy od szybkości wykonywania poszczególnych elementarnych operacji arytmetycznych i logicznych. Dokładność tych komputerów nie jest w żaden sposób uzależniona od dokładności wykonania poszczególnych elementów, lecz zależy jedynie od liczby pozycji przeznaczonych do zapisania liczby w komputerze – długości słowa maszynowego.

Oprócz dużej szybkości i dokładności obliczeń wynikających z przyjętej struktury sprzętowej najważniejszą zaletą komputerów cyfrowych jest ich uniwersalność wyrażana przez oprogramowanie. W przypadku komputerów cyfrowych zmiana przeznaczenia polega na zmianie programu i dostarczeniu nowych danych bez potrzeby jakiejkolwiek ingerencji w strukturą sprzętową.

Schemat blokowy komputera cyfrowego

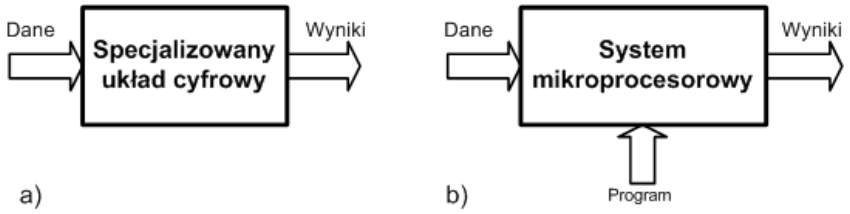
Współczesne komputery cyfrowe wywodzą się z koncepcji komputera zaproponowanego w połowie lat 40 w USA przez węgierskiego matematyka **Johna von Neumanna** współpracującego z zespołem naukowców z Uniwersytetu w Pensylwanii. W roku 1945 von Neuman opublikował raport, w którym opisał projekt uniwersalnego komputera, znanego pod nazwą **maszyna z Princeton**.

W komputerze von Neumanna (rysunek 2.1) można wyróżnić następujące elementy: **pamięć** złożoną z elementów przyjmujących stan 0 lub 1, **arytmometr** zdolny wykonywać działania arytmetyczne, logiczne i inne, sterowanie zapewniające poprawne wykonanie każdego rozkazu, **wejście** umożliwiające wprowadzanie programu i danych oraz **wyjście** pozwalające na wyprowadzanie wyników przetwarzania. Program, czyli zbiór instrukcji, według których mają odbywać się obliczenia, jest wpisywany do pamięci. Kolejne rozkazy programu są pobierane przez jednostkę sterującą komputerem w takt centralnego zegara i rozpoznawane zgodnie z mikroprogramem wpisanym w układ elektroniczny.



Rysunek 2.1. Architektura komputera von Neuman'a

Podkreślmy, że program jest przechowywany w pamięci komputera i jego działanie może zmieniać zawartość dowolnego obszaru pamięci (programy mogą się także same modyfikować). Fizycznie nie ma żadnej różnicy między danymi i programami przechowywanymi w pamięci komputera: są podobnie kodowane jako ciąg zer i jedynek i tak samo zrealizowane technicznie. Można, więc powiedzieć, że celem działania tego typu komputera jest przejście w takt zegara od jednego stanu zawartości pamięci (danego na początku) do innego, zawierającego oczekiwany wynik. Kluczowym punktem komputera działającego wg. tej koncepcji (znanego pod nazwą maszyna Neumanna) była możliwość modyfikacji własnego programu.

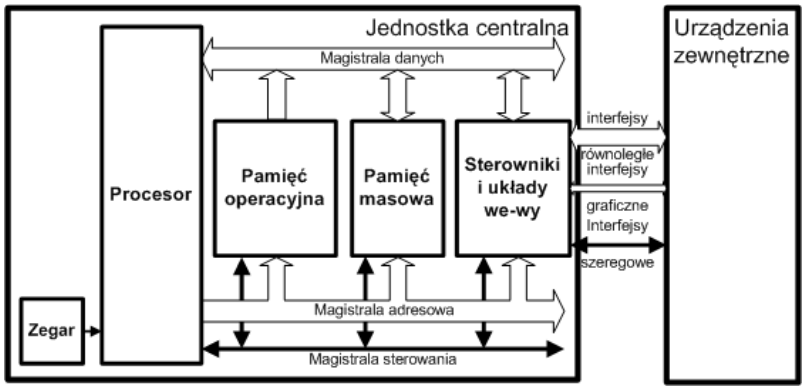


Rysunek 2.2. Architektury komputerów: a) specjalizowany układ cyfrowy, b) system mikroprocesorowy

Architektura obecnie konstruowanych komputerów jest bardziej złożona. Współczesny komputer [47] może być zrealizowany w postaci specjalizowanego układu cyfrowego lub systemu mikroprocesorowego. W pierwszym przypadku (rysunek 2.2a) przetwarzanie informacji zależy wyłącznie od użytych układów cyfrowych i sposobu ich podłączenia, czyli od sprzętu (ang. *hardware*). Natomiast w drugim przypadku (rysunek 2.2b) najważniejszym elementem jest uniwersalny układ przetwarzający informację – system mikroprocesorowy. Do systemu mikroprocesorowe-

go oprócz danych wejściowych musimy więc dostarczyć także program lub zestaw programów, czyli oprogramowanie (ang. *software*). Takie rozwiązanie jest stosowane wszędzie tam gdzie zachodzi potrzeba zmiany sposobu przetwarzania informacji: komputery powszechnego użytku, komputery przemysłowe, komputery pokładowe, sterowniki programowalne, itd.

W latach siedemdziesiątych i osiemdziesiątych komputery dzielono na „duże", „średnie" i „małe". Obecnie ten tradycyjny podział komputerów już nie obowiązuje a głównym powodem tego stanu jest dynamiczny rozwój mikroelektroniki i techniki mikroprocesorowej. Rozwój ten wykazał, że o mocy obliczeniowej komputera decyduje liczba podstawowych operacji wykonywanych w jednostce czasu, długość słowa wyrażającego dane oraz pojemność pamięci wewnętrznej komputera. Wymienione parametry nie są bezpośrednio związane z gabarytami urządzenia, w szczególności, gdy porównujemy komputery wykonane za pomocą różnych technologii, zaliczane do różnych generacji (0 - na przekaźnikach; 1 – na lampach elektronowych; 2 – na tranzystorach; 3 – na układach scalonych małej skali integracji).



Rysunek 2.3. Schemat blokowy współczesnego komputera

Współczesne komputery (rysunek 2.3), nazywa się mikrokomputerami i zalicza się do czwartej generacji budowanej w oparciu o układy scalone dużej i wielkiej skali integracji. Typowy mikrokomputer [47, 48] zawiera jednostkę centralną zbudowaną z procesora, pamięci i układów wejścia - wyjścia połączonych ze sobą za pomocą magistrali danych, adresowej i sterującej, współpracującą z wieloma urządzeniami zewnętrznymi dołączonymi za pośrednictwem różnorodnych interfejsów.

Procesor komputera

Procesorem [32, 47] nazywa się urządzenie dokonujące bezpośredniego przetwarzania danych i sterujące tym procesem w sposób programowy. Tworzą go połączone razem: jednostka arytmetyczno-logiczna i sterowanie komputera, a charakteryzuje zbiór operacji elementarnych, tj. operacji, które mogą być wykonane na żądanie. Wyrażenie operacja elementarna oznacza, iż z naszego punktu widzenia operacja stanowi całość, tzn. nie interesuje nas, jakie akcje należy kolejno podjąć, aby dobiegła końca. Wymagamy natomiast, aby każda operacja elementarna została całkowicie zrealizowana w skończonym czasie.

Procesor jest najważniejszą częścią mikrokomputera, gdyż jest to jedyny układ w sposób aktywny przetwarzający dane. W literaturze angielskiej procesor nosi nazwę CPU (ang. *Central Processing Unit*) - centralnej jednostki przetwarzającej komputera. Rolę procesora może pełnić mikroprocesor lub mikrokontroler.

Pamięć komputera

Pamięć w komputerze [32, 47] służy do przechowywania informacji. Podstawową jednostką pamięci jest komórka. Jest to najmniejszy element, w którym informacja może być zapisana, zapamiętana i z którego może być odczytywana bez zmiany treści zapamiętanej informacji. Całkowita liczba komórek określa pojemność pamięci. Komórki grupowane są w słowa. Z każdą komórką (słowem) związany jest zatem jej adres oraz zawartość. Adresem komórki jest liczba w postaci binarnej, której przyporządkowana jest komórka. Zawartość komórki charakteryzuje się stanami logicznymi 1 lub 0. Pojemność pamięci mierzy się w bitach, bajtach (1 bajt = 8 bitów) lub słowach. Powszechnie stosuje się oznaczenia: **b** na bit oraz **B** na bajt, natomiast na słowa nie stosuje się oznaczeń. Pojemność pamięci komputera wyraża się często z wykorzystaniem mnożników określających krotność jednostek: 1 **K** (1 kilo) oznacza 2^{10}=1024; 1 **M** (1 mega) oznacza 2^{20}=1048576; 1 **G** (1 giga) oznacza 2^{30}=1073741824. Oprócz pojemności do podstawowych parametrów pamięci zalicza się jej szybkość wyrażaną za pomocą zależności czasowych. Powszechnie używane są dwa czasy:

- czas dostępu – mierzony od chwili podania adresu do ukazania się informacji na wyjściu pamięci;
- czas cyklu – mierzony od chwili pojawienia się żądania dostępu do pamięci do momentu gotowości na przyjęcie następnego żądania dostępu.

Ważnym parametrem jest również organizacja pamięci, czyli sposób podziału obszaru pamięci na słowa. Dla przykładu moduł pamięci o pojemności 8 Kbitów może być zorganizowany jako:

- 1 Kx8 bitów, co oznacza 1024 słowa ośmiobitowe;
- 2 Kx4 bity, co oznacza 2048 słowa czterobitowe;
- 8 Kx1 bit, co oznacza 8192 pojedyncze bity.

Pamięć współczesnego komputera nie stanowi zazwyczaj jednej całości. Najczęściej występuje w postaci układu hierarchicznego, gdzie każdy z poziomów hierarchii charakteryzuje określona szybkość działania, pojemność i koszt. Ze względu na realizowane funkcje pamięć dzieli się na główną i pomocniczą.

Pamięcią główną jest pamięć operacyjna współpracująca „na bieżąco" z procesorem, w której przechowywane są aktualnie wykonywane programy i argumenty bieżących operacji arytmetyczno-logicznych. Jest to z reguły szybka pamięć o dość małej pojemności.

Pamięcią pomocniczą jest pamięć masowa, przechowująca programy i dane, z których procesor aktualnie nie korzysta. Jest to pomięć powolna, najczęściej zewnętrzna, lecz o bardzo dużej pojemności.

Układy i sterowniki wejścia - wyjścia

Układy wejścia-wyjścia [32, 47] są układami elektronicznymi pośredniczącymi w wymianie informacji pomiędzy systemem mikroprocesorowym a współpracującymi z tym systemem urządzeniami zewnętrznymi nazywanymi urządzeniami peryferyjnymi.

Współpraca urządzeń zewnętrznych z jednostką centralną mikrokomputera jest możliwa jedynie po odpowiednim dopasowaniu parametrów elektrycznych, zdefiniowaniu znaczenia poszczególnych linii łączących oraz określeniu sposobu transmisji, np. przebiegów czasowych sygnałów, ich wzajemnych zależności.

Oprócz magistrali danych, adresowej i sterującej istniejących tradycyjnie w każdym systemie mikroprocesorowym współczesne komputery wyposaża się w dodatkowe magistrale określonego standardu umożliwiające rozszerzanie możliwości czy rozbudowę komputera. Do najbardziej znanych standardów magistral rozszerzających należy zaliczyć następujące magistrale: **ISA** lub **AT BUS** - (ang. *Industry Standard Architecture*) 16 bitowa magistrala danych stosowana w komputerach klasy PC, **PCI** -

(ang. *Peripheral Component Interconnect*) 32 bitowa szyna systemowa PCI32 spełniająca normy standardu Plug & Play, wprowadzona w 1993r. przez firmę Intel i unowocześniona w 2000 r. do wersji 64 bitowej PCI64, **AGP** - (ang. *Accelerated Graphics Port*) interfejs komunikacyjny mający na celu zwiększenie przepustowości kart graficznych o przepustowości: 266 MB/s, 533 MB/s i 1066 MB/s.

Urządzenia zewnętrzne

Urządzenia zewnętrzne komputera [6, 48] to urządzenia służące do wprowadzania i wyprowadzania informacji do/z komputera przy udziale właściwego sterownika. Do tej grupy urządzeń peryferyjnych należą:

Klawiatury – podstawowe urządzenie umożliwiające komunikację operatora z komputerem, najczęściej wzorowane na maszynie do pisania, stanowi tablicę, na której rozmieszczone są przyciski oznaczone: literami, cyframi oraz specjalnymi znakami, a także tzw. klawisze funkcyjne umożliwiające wykonanie specyficznych czynności. Istnieje wiele typów klawiatur, różniących się m.in. układem klawiszy czy zestawem klawiszy funkcyjnych;

Drukarki i plotery – komputerowe urządzenie peryferyjne, używane do drukowania tekstów i grafiki na papierze. Drukarki i plotery różnią się szybkością, jakością i techniką druku. Ze względu na technikę wydruku istnieje kilka rodzajów ploterów i drukarek: igłowe, termiczne, atramentowe, laserowe;

Skanery – wejściowe urządzenia zewnętrzne przetwarzające dowolne obrazy (fotografie, teksty, rysunki) z postaci analogowej na cyfrową;

Pamięci zewnętrzne – urządzenia służące do trwałego magazynowania danych. Zaliczamy do nich taśmy magnetyczne, dyski magnetyczne stałe i wymienne, dyski optyczne;

Manipulatory – urządzenia ułatwiające dostęp operatora do komputera. Do tej grupy urządzeń zalicza się drążki, myszki, kule i potencjometry, pióra świetlne, ekrany dotykowe;

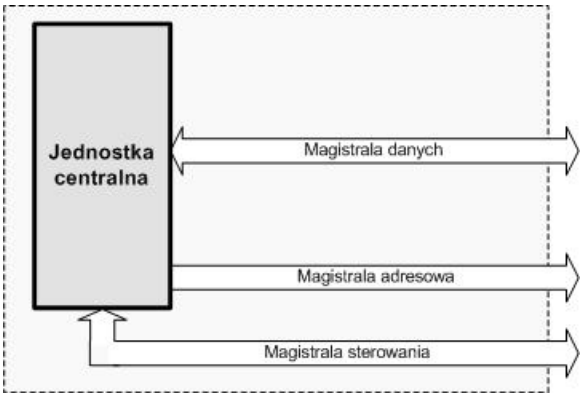
Monitory – urządzenia stosowane do wyprowadzania i zobrazowania informacji. Obecnie dominują kolorowe monitory budowane w oparciu o wyświetlacze ciekłokrystaliczne LCD (ang. - *Liquid Crystal Display*), które całkowicie wyparły stosowane wcześniej monitory z klasycznymi kineskopami CRT (ang. - *Cathode Ray Tube*).

2.2. Budowa, cykl rozkazowy, podstawowe operacje arytmetyczno-logiczne, lista rozkazów typowego mikrokontrolera

Najważniejszymi układami scalonymi techniki mikroprocesorowej są te elementy, które mają możliwość przetwarzania danych. Takim elementem jest procesor, czyli centralna jednostka przetwarzająca CPU. Od końca lat 70 produkowane są dwie podstawowe grupy układów zawierające w swojej strukturze CPU. Są to:

- mikroprocesory (ang. *microprocessor*);
- mikrokontrolery (ang. *microcontroller*).

Mikroprocesor (rysunek 2.4.) to układ scalony w strukturze którego zintegrowana jest kompletna jednostka centralna komputera [18, 33, 36]. Mikroprocesor komunikuje się z otoczeniem przy pomocy trzech zespołów sygnałów cyfrowych, tzw. szyn (ang. *Bus*).

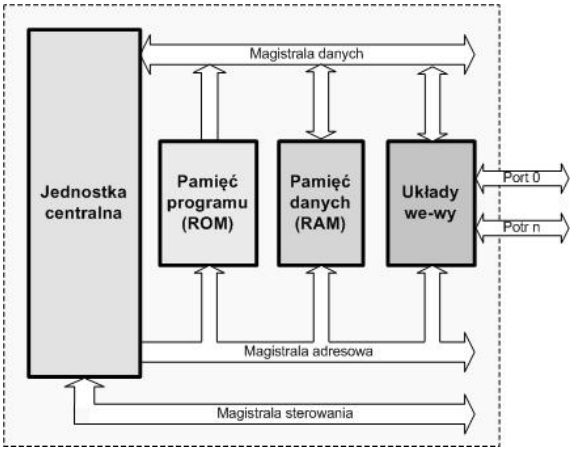


Rysunek 2.4. Budowa mikroprocesora

Jest to szyna danych (ang. *Data Bus*), szyna adresowa (ang. *Address Bus*) oraz szyna sterująca (ang. *Control Bus*). Zespól tych trzech szyn określany jest jako szyna systemowa. Przy pomocy tych szyn przyłą-

czane są do mikroprocesora dwa pozostałe elementy niezbędne do budowy komputera: pamięć oraz układy peryferyjne.

Mikrokontroler - (rysunek 2.5.) to układ scalony w strukturze którego zintegrowane są wszystkie elementy kompletnego komputera: jednostka centralna, pamięć oraz układy wejścia-wyjścia. Jest to zatem komputer w jednym układzie scalonym [10, 12, 38]. Mikrokontroler komunikuje się z otoczeniem za pośrednictwem układów wejścia-wyjścia.



Rysunek 2.5. Budowa mikrokontrolera

Przedstawione definicje mikroprocesora i mikrokontrolera mogły by sugerować, że mikrokontroler jest układem scalonym o wyższym stopniu zaawansowania technicznego, ponieważ zawiera on w swojej strukturze kompletny komputer, podczas gdy mikroprocesor tylko jego część. Tak nie jest. Są to najważniejsze i równorzędne układy scalone na których bazuje współczesna technika mikroprocesorowa.

Mikroprocesory ukierunkowane są na zastosowanie w technice obliczeniowej. Najbardziej masowo stosowane są obecnie w komputerach osobistych. Mikroprocesory mają zapewnić możliwie dużą moc obliczeniową, tj. dużą szybkość przetwarzania danych. Osiąga się to poprzez złożoną architekturę wewnętrzną jednostki centralnej (np. architektura typu RISC), dużą szerokość szyny danych (obecnie najczęściej 32 bity) oraz dużą częstotliwość zewnętrznego zegara taktującego pracę układu (obecnie ponad 3000 MHz). Aktualny w danym momencie stan rozwoju mikroelektroniki wyraża się m.in. liczbą tranzystorów, które daje się upakować w strukturze pojedynczego układu scalonego. Projektanci mikroprocesorów świadomie zakładają, że całkowite możliwości wynikające z rozwoju mikroelektroniki zostaną wykorzystane na zwiększenie

mocy obliczeniowej jednostki centralnej w postaci jednego układu scalonego, zaś pozostałe elementy komputera (tj. pamięć i układy wejścia-wyjścia) - dla których nie ma już miejsca w strukturze układu scalonego – zostaną przyłączone do mikroprocesora jako osobne zewnętrzne układy scalone.

Mikrokontrolery ukierunkowane są na zastosowanie w układach sterowania. W wielu zastosowaniach tego rodzaju niezbędną do realizacji zadania moc obliczeniową jednostki centralnej daje się określić. Nie obowiązuje przy tym zasada, że im większa moc obliczeniowa jednostki centralnej tym lepiej. Przykładem niech będzie zastosowanie mikrokontrolera do obsługi klawiatury. Zadaniem komputera jest w tym przypadku rozpoznawanie, który przycisk klawiatury naciśnięty został przez użytkownika. Zapotrzebowanie na moc obliczeniową w przypadku tego zadania określone jest przez maksymalną prędkość naciskania przycisków przez człowieka. Z praktyki wynika, że wystarczy sprawdzać stan przycisków klawiatury z częstotliwością ok. 10 razy na sekundę. Zastosowanie zatem do tego zadania mikrokontrolera, który będzie w stanie sprawdzać stan klawiatury z częstotliwością np. 1000 razy na sekundę mija się z celem, gdyż nie spowoduje to lepszego wykonania zadania. Klawiatura jest z reguły częścią większego urządzenia np. regulatora temperatury. W takim przypadku zamiast dużej mocy obliczeniowej bardziej celowe byłoby umieszczenie w strukturze układu scalonego z centralną jednostką obliczeniową dodatkowych układów wejścia-wyjścia, np. takich które umożliwią bezpośrednie przyłączenie przycisków klawiatury do wyprowadzeń układu scalonego bez konieczności stosowania dodatkowych układów pośredniczących pomiędzy jednostką centralną a klawiaturą. Mikrokontroler jest zatem układem scalonym, którego stosowanie minimalizuje liczbę układów niezbędnych do realizacji danego zadania.

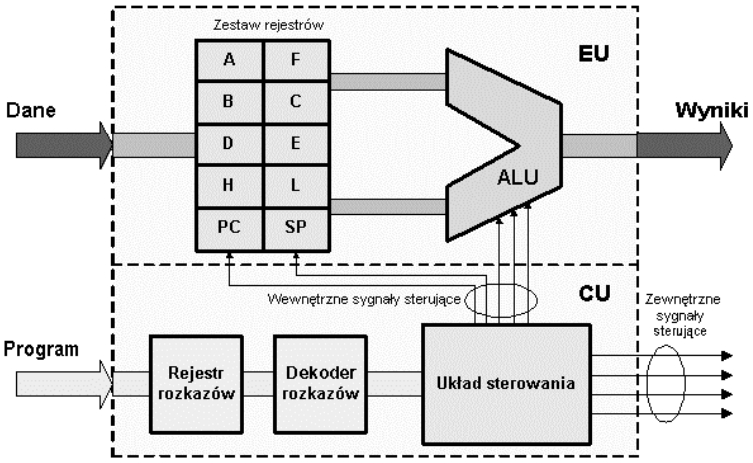
Jest jeszcze jedna istotna różnica pomiędzy mikroprocesorem a mikrokontrolerem. Zadania, które wykonuje komputer stosowany w układach sterowania są zazwyczaj niezbyt złożone pod względem obliczeniowym, wymagają natomiast często szybkiej reakcji na sygnały zewnętrzne (np. szybkiej reakcji na informację o przekroczeniu wartości granicznej określonej wielkości). Funkcję tą realizuje najlepiej przyporządkowany jednostce centralnej tzw. układ przerwań (ang. *Interrupt Unit*). W mikrokontrolerach jest on zazwyczaj szybszy i bardziej rozbudowany niż w mikroprocesorach.

W praktyce szczegółowa znajomość budowy wewnętrznej i sposobu działania jednostki centralnej nie jest konieczna. Dotyczy to zarówno projektanta systemów mikroprocesorowych od strony sprzętowej jak

i programisty tych systemów. Dlatego w dalszej części rozdziału zostanie przedstawiony tylko krótki opis działania jednostki centralnej niezbędny dla zrozumienia funkcjonowania systemu mikroprocesorowego.

Budowa mikroprocesora

Poszczególne mikroprocesory [10, 33, 36] różnią się dość znacznie strukturą logiczną - organizacją. W każdym z nich występują jednak podzespoły wynikające logicznie z zadań, jakie pełni mikroprocesor. Na schemacie przedstawionym na rysunku 2.6 pokazano główne bloki funkcjonalne mikroprocesora: jednostkę wykonawczą EU (ang. *Execution Unit*) i jednostkę sterującą CU (ang. *Control Unit*).



Rysunek 2.6. Schemat blokowy mikroprocesora

Zadaniem jednostki wykonawczej jest przetwarzanie informacji, czyli wykonywanie wszelkich operacji arytmetycznych i logicznych. Rodzaj wykonywanych operacji zależy od wewnętrznych sygnałów sterujących wytwarzanych przez jednostkę sterującą.

W skład jednostki wykonawczej wchodzi jednostka arytmetyczno-logiczna ALU (ang. *Arithmetic-Logic Unit*) oraz zestaw współpracujących z nią rejestrów. Informacją wejściową części wykonawczej są dane, zaś wyjściową wyniki (liczby, informacja tekstowa, sygnały sterujące pracą urządzeń, itp.).

W skład jednostki sterującej wchodzą: rejestr rozkazów IR (ang. *instruction register*), dekoder rozkazów i układ sterowania. W rejestrze rozkazów przechowywany jest pobrany wcześniej z pamięci kod aktualnie wykonywanego rozkazu, który jest dekodowany w dekoderze rozkazów.

Po określeniu rodzaju rozkazu układ sterowania wytwarza wewnętrzne i/lub zewnętrzne sygnały sterujące, realizujące dany rozkaz.

Rejestry mikroprocesora

Zarówno jednostka arytmetyczno-logiczna jak i układ sterowania współpracują z określonym zestawem rejestrów. Zawartość pewnej części rejestrów z tego zestawu może być zmieniana w wyniku wykonania przez procesor określonej instrukcji. Rejestry takie nazywamy rejestrami dostępnymi programowo. Pozostałe rejestry są niedostępne dla użytkownika i ich zestaw nie jest zwykle znany. W rejestrach dostępnych programowo występują takie typy rejestrów, których odpowiedniki znajdują się praktycznie w każdym procesorze. Ich pojemność czy ilość może się zmieniać, jednak wykonywane zadania pozostają takie same.

Zdecydowana większość mikroprocesorów zawiera następujące rejestry dostępne programowo:

- akumulator A (ang. *accumulator*);
- rejestr stanu SF (ang. *status flag*);
- zestaw rejestrów uniwersalnych Ri (ang. *general purpose registers*);
- licznik rozkazów PC (ang. *program counter*);
- wskaźnik stosu SP (ang. *stack pointer*).

Akumulator jest jednym z najważniejszych dla użytkownika rejestrów uniwersalnych mikroprocesora. Zwykle zawiera on jeden z argumentów wykonywanej operacji i jest miejscem, do którego jest ładowany wynik wykonywanej operacji. Długość (liczba bitów) akumulatora i rejestrów specjalnych wyznacza podstawowy parametr mikroprocesora – długość słowa.

Rejestrem stanu nazywamy rejestr zawierający dodatkowe cechy wyniku wykonywanej operacji, niezbędne do podjęcia decyzji o dalszym sposobie przetwarzania informacji. Cechami tymi mogą być przykładowo znak wyniku, wystąpienie przekroczenia zakresu czy parzystość (parzysta bądź nieparzysta ilość jedynek w wyniku). Wystąpienie określonego zdarzenia, na przykład wyniku dodatniego bądź ujemnego sygnalizowane jest ustawieniem lub wyzerowaniem określonego bitu w rejestrze stanu. Ustawiane bity nazywane są znacznikami (stąd inna nazwa tego

rejestru to rejestr znaczników) lub flagami. Flagi mogą być używane przy tworzeniu rozgałęzień w programie.

Licznik rozkazów jest jednym z istotniejszych rejestrów umożliwiających zrozumienie działania mikroprocesora. W nowszych mikroprocesorach nosi on nazwę wskaźnika instrukcji IP (ang. *Instruction Pointer*), co jest chyba bardziej trafne. Ostatecznie **licznikiem rozkazów** nazywamy rejestr mikroprocesora zawierający adres komórki pamięci, w której przechowywany jest kod rozkazu przeznaczonego do wykonania jako następny.

Z powyższej definicji wynika, że po wczytaniu kolejnego kodu rozkazu zawartość licznika rozkazów powinna zostać zmieniona tak, aby wskazywał on na kolejny rozkaz przeznaczony do wczytania do mikroprocesora.

Ponieważ w przeważającej części program jest wykonywany kolejno, instrukcja po instrukcji, to pobranie kodu rozkazu z kolejnej komórki pamięci powoduje zwiększenie zawartości licznika rozkazów o 1. W pewnych przypadkach zawartość licznika rozkazów jest zmieniana w wyniku wykonania określonego rozkazu, dlatego jest to rejestr dostępny programowo.

Wskaźnik stosu służy do adresowania wydzielonego obszaru pamięci, w którym adresy lub dane są zapisywane i odczytywane zgodnie z regułą pamięci LIFO (ang. *Last in First Out*), co oznacza, że kolejność odczytu słów z pamięci jest odwrotna niż kolejność ich zapisywania. Tak zorganizowaną pamięć nazywa się stosem. Natomiast **wskaźnikiem stosu** nazywamy rejestr zawierający adres ostatniej zapełnionej komórki stosu (wierzchołka stosu).

W zależności od typu mikroprocesora stos może budować się w kierunku rosnących adresów, czyli „do góry", lub w kierunku adresów malejących. Z definicji wskaźnika stosu wynika, że przy założeniu narastania stosu w kierunku adresów rosnących każdy zapis na stos zwiększa zawartość wskaźnika stosu, a każdy odczyt zmniejsza jego zawartość. Dzięki temu wskaźnik stosu „wskazuje" zawsze na ostatnią zapełnioną komórkę stosu.

Jednym z klasycznych zastosowań stosu jest zapamiętanie adresu powrotu do programu wywołującego w przypadku wywołania tak zwanego podprogramu. Ponieważ podprogram może z kolei wywoływać inny podprogram, adresy powrotów odkładane są na stos, gdyż muszą być odczytywane w kolejności odwrotnej do kolejności ich zapisu.

Oprócz wymienionych wyżej rejestrów, wykonujących ściśle określone zadania, każdy mikroprocesor dysponuje pewnym zestawem rejestrów zwanych rejestrami roboczymi lub rejestrami ogólnego przeznaczenia. Rejestry takie mogą przechowywać argumenty wykonywanych operacji, wyniki lub ich adresy. Rejestry te mogą również pełnić pewne dodatkowe funkcje przewidziane przez projektanta mikroprocesora, na przykład liczników ilości wykonywanych pętli. Przykładem rejestrów roboczych w mikroprocesorach AVR są 32 rejestry ogólnego przeznaczenia natomiast w procesorze rodziny 8051 wyróżnia się cztery banki rejestrów oznaczonych symbolami R0÷R7 pełniące funkcje rejestrów roboczych.

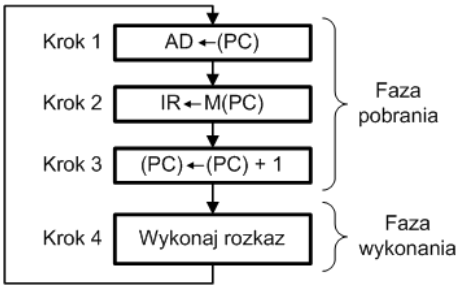
Cykl rozkazowy mikroprocesora

Jednostka CPU mikroprocesora należy do grupy układów cyfrowych określanych jako układy synchroniczne i sekwencyjne. Synchroniczność oznacza, że wszystkie operacje wykonywane przez jednostkę centralną odbywają się w rytmie narzuconym przez zewnętrzny układ generujący sygnał periodyczny o stałej częstotliwości. Sygnał ten nosi nazwę **sygnału zegarowego** (ang. *clock*). Impulsy zegarowe wyznaczają początek i koniec wszystkich elementarnych działań wykonywanych przez mikroprocesor. Okres impulsów zegarowych nazywany jest **taktem zegarowym**. Sekwencyjność oznacza, że jednostka centralna posiada własną pamięć potrzebną np. do przechowywania argumentów rozkazów niezbędnych do wykonania na nich określonej operacji.

Realizując program mikroprocesor wykonuje więc pewne powtarzające się czynności polegające na cyklicznym pobieraniu kodów rozkazów z pamięci i wczytywaniu ich do układu sterowania mikroprocesora, a następnie realizacji rozkazu, którego kod został pobrany. Czas niezbędny do wykonania rozkazu nosi nazwę **cyklu rozkazowego**. W cyklu tym możemy wyróżnić dwie fazy zwane: **fazą pobrania** kodu rozkazu (ang. *fetach*) i **fazą wykonania** rozkazu (ang. *execution*). Następstwo kolejnych faz przedstawia rysunek 2.7.

Faza pobrania polega na odczytaniu kodu rozkazu z komórki pamięci o adresie przechowywanym w liczniku rozkazów, a następnie na umieszczeniu tego kodu w rejestrze rozkazów IR znajdującym w układzie sterowania mikroprocesora. Kod rozkazu przesyłany jest do mikroprocesora magistralą danych. Następnie zawartość licznika rozkazów jest modyfikowana tak, aby wskazywał on na adres następnej komórki pamięci programu zawierającej kolejny rozkaz przeznaczony do pobrania. Po zakończeniu fazy pobrania następuje faza wykonania. Rozpoczyna się ona od chwili umieszczenia kodu rozkazu w rejestrze rozkazów. Polega na

zdekodowaniu kodu rozkazu znajdującego się w rejestrze rozkazów, czyli stwierdzeniu, jaki to rozkaz. Na tej podstawie układ sterowania wytwarza wewnętrzne i/lub zewnętrzne sygnały sterujące realizujące dany rozkaz.



Rysunek 2.7. Fazy cyklu rozkazowego

Każdy cykl rozkazowy składa się z kilku cykli maszynowych. Przez **cykl maszynowy** rozumie się część cyklu rozkazowego związaną z odwoływaniem się do pamięci lub układów wejścia-wyjścia. Cykl maszynowy składa się na ogół z kilku taktów zegarowych. Faza pobrania cyklu rozkazowego zawiera więc tyle cykli maszynowych, ile słów zawiera rozkaz. Faza wykonania może zawierać różną liczbę cykli maszynowych, zależnie od rodzaju wykonywanego rozkazu. Kolejne etapy realizacji fazy pobierania i fazy wykonania rozkazu można przedstawić następująco:

Faza pobrania

1. Adresowanie: podanie zawartości licznika rozkazów na magistralę adresową AD ← (PC)
2. Wczytanie zawartości zaadresowanej komórki pamięci do rejestru rozkazów mikroprocesora: IR ← M(PC)
3. Zwiększenie zawartości licznika rozkazów: (PC) ← (PC) + 1

Faza wykonania

4. Zdekodowanie kodu rozkazu i wytworzenie sygnałów sterujących realizujących dany rozkaz.

Jak pokazano na rysunku 2.7 w cyklu rozkazowym następują po sobie na przemian faza pobrania i faza wykonania. W celu przyspieszenia pracy mikroprocesora stosuje się różne modyfikacje cyklu rozkazowego. Jedna z metod, zwana prefetchingiem, polega na równoległym wykonywaniu

fazy pobrania następnego rozkazu, jeszcze w trakcie realizacji fazy wykonania rozkazu poprzedniego. Rozwinięciem idei prefetchingu jest praca potokowa, polegająca na równoległym wykonywaniu kilku faz realizacji rozkazu.

Lista rozkazów mikroprocesora

Wykonywane przez dany mikroprocesor operacje przetwarzania danych należą do zbioru rozkazów zwanego **listą rozkazów** i tworzą zbiór funkcjonalnie pełny, co oznacza, że za pomocą tych operacji można realizować dowolnie złożony algorytm przetwarzania informacji. Listy rozkazów produkowanych obecnie mikroprocesorów są bardzo zróżnicowane zarówno pod względem rodzaju rozkazów jak i ich liczby. Wpływa to bezpośrednio na architekturę mikroprocesora. Słowo „architektura" ilustruje przyjęty poziom opisu, skoncentrowany na przedstawieniu ogólnej budowy i funkcji oferowanych przez mikroprocesory.

W dużym uproszczeniu mikroprocesory o architekturze CISC (ang. *Complex Instruction Set Computer*) charakteryzują się rozbudowaną listą rozkazów (często powyżej 100 rozkazów) oraz dużym stopniem złożoności elementarnych operacji wykonywanych w ramach poszczególnych rozkazów. Ponadto wykonywanie rozkazów w procesorze o architekturze CISC trwa stosunkowo długo.

W nowszych mikroprocesorach stosuje się architekturę RISC (ang. *Reduced Instruction Set Computer*). Lista rozkazów mikroprocesora o tej architekturze obejmuje zazwyczaj tylko kilkadziesiąt prostych rozkazów natomiast ich wykonywanie trwa znacznie krócej niż w mikroprocesorze o architekturze CISC. Również **moc obliczeniowa** mikroprocesorów o architekturze RISC wyrażana w jednostkach MIPS (ang. *Million Instructions Per Second*), definiowana jako liczba milionów rozkazów wykonywanych przez mikroprocesor w czasie 1 sekundy jest z reguły większa od prędkości przetwarzania procesorów z architekturą CISC.

Ostatecznie, rozkazem (instrukcją maszynową) nazywamy najprostszą operację, której wykonania programista może zażądać od procesora.

Sposób realizacji rozkazu nie jest istotny dla użytkownika systemu i z reguły nie jest znany. Nawet jeżeli wiemy, że dany rozkaz jest realizowany jako ciąg prostszych operacji, zwanych mikrooperacjami czy mikrorozkazami, i tak nie mamy żadnego wpływu na sposób realizacji rozkazu.

Rozkazy tworzące listę rozkazów możemy podzielić na kilka podstawowych grup. W zależności od realizowanych funkcji rozróżniamy:

1. rozkazy przesłań – bitów, słów, par słów;
2. rozkazy arytmetyczne i logiczne – dodawanie, odejmowanie, sumowanie logiczne, iloczyn logiczny, przesunięcie, itp.;
3. rozkazy sterujące (skoki, wywołania podprogramów, pętle itp.);
4. inne (rozkazy komunikacji z urządzeniami zewnętrznymi, rozkazy testujące itp.).

Rozkazy przesłań są najczęściej wykonywanymi rozkazami. Nie zmieniają one wartości informacji, natomiast przenoszą ją z miejsca na miejsce. Ich duża częstotliwość wykonywania jest dość oczywista. Jeśli chcemy wykonać jakąś operację, musimy zwykle pobrać jej argumenty, a po jej wykonaniu zapisać wynik. Wśród przesłań wyróżnia się czasami operacje na stosie oraz instrukcje wejścia/wyjścia, przesyłające lub odczytujące dane z portów wejścia/wyjścia.

Rozkazy arytmetyczne i logiczne służą do przetwarzania informacji, czyli w wyniku ich wykonania jest ona zmieniana. Do tych rozkazów należą rozkazy wykonujące podstawowe działania arytmetyczne i logiczne oraz rozkazy cyklicznego przesuwania informacji, rozkazy porównań itp.

Rozkazy sterujące stanowią grupę rozkazów pozwalającą zmienić kolejność wykonywania instrukcji programu. Należą do nich przykładowo rozkazy skoków bezwarunkowych i warunkowych (warunkiem w takim skoku jest wartość określonej flagi), bezwarunkowe i warunkowe wywołania podprogramów czy też instrukcje pętli (czyli instrukcje powodujące kilkakrotne powtórzenie pewnego ciągu instrukcji).

Pozostałe instrukcje to zwykle instrukcje charakterystyczne dla danego typu procesora.

3 Organizacja i zasada działania systemu mikroprocesorowego

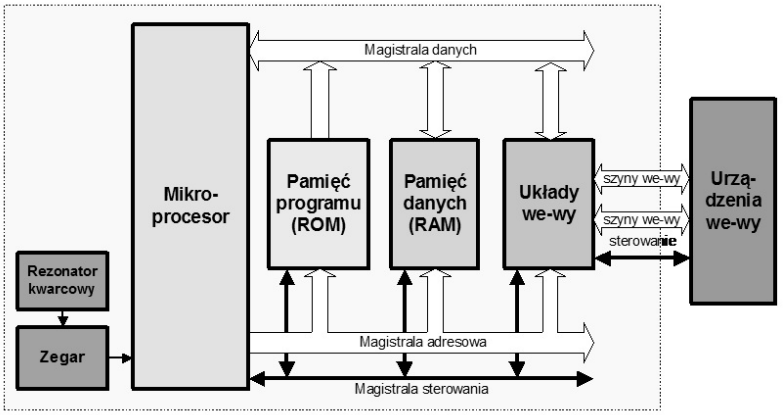
W tym rozdziale

- Budowa systemu mikroprocesorowego
- Rodzaje oraz przeznaczenie pamięci programu i danych
- Układy wejścia-wyjścia i podstawowe układy peryferyjne
- Interfejsy i sterowniki do szeregowej transmisji danych
- Urządzenia wejściowe i wyjściowe maszyn i urządzeń mechatronicznych

3.1. Budowa systemu mikroprocesorowego

Typowy system mikroprocesorowy [10, 20, 31] (rysunek 3.1), czy to realizowany przy użyciu mikroprocesora, czy mikrokontrolera, składa się z **mikroprocesora**, jego **otoczenia** i **oprogramowania**, gdzie:

- **mikroprocesor**, to procesor wykonany jako układ wielkiej lub bardzo skali integracji;
- **otoczeniem** są elementy, układy i urządzenia niezbędne do konfiguracji układowej wokół procesora pozwalające zrealizować problem opisany programem;
- **oprogramowanie** stanowią programy użytkowe realizowane w danej konfiguracji oraz programy systemowe ułatwiające tworzenie i eksploatację programów użytkowych.



Rysunek 3.1. Schemat blokowy systemu mikroprocesorowego

Elementem podłączanym bezpośrednio do mikroprocesora jest generator impulsów zegarowych. Większość mikroprocesorów posiada wbudowany generator i wtedy podłącza się tylko rezonator kwarcowy lub ceramiczny. Bezpośrednio do mikroprocesora podłączany jest również układ wytwarzający sygnał zerowania (ang. *RESET*).

Wszystkie bloki funkcjonalne połączone są wieloma liniami (przewodami) stanowiącymi drogi dla adresów, sygnałów sterujących, rozkazów i danych. Linie te nazywane są magistralami lub szyną systemową. Adres podany na magistralę adresową określa, z którym elementem otoczenia mikroprocesora ma nastąpić wymiana danych. Sygnały sterujące podawane magistralą sterowania określają, w którym momencie i w jakim kierunku maja być przesyłane dane. Dane są przesyłane magistralą danych. Do szyny systemowej podłączone są, zależnie od potrzeb, układy otoczenia mikroprocesora, do których zaliczamy:

- pamięć programu;
- pamięć danych;
- układy wejść-wyjść (porty, układy czasowe, sterowniki transmisji szeregowej, przetworniki A/C i C/A, zegary czasu rzeczywistego).

W skład systemu mikroprocesorowego wchodzą również różne urządzenia, które nie są łączone bezpośrednio przez szynę systemową. Do ich podłączenia wykorzystuje się układy wejść-wyjść. Są to takie urządzenia, jak:

- klawiatury;
- wyświetlacze;
- sygnalizatory;
- czujniki;
- silniki elektryczne;
- przekaźniki;
- zawory elektromagnetyczne, itp.

3.2. Rodzaje oraz przeznaczenie pamięci programu i danych

Jako **pamięci operacyjne** systemów mikroprocesorowych stosowane są **pamięci półprzewodnikowe o dostępie bezpośrednim** (swobodnym) [18, 22, 37]. Ze względu na pełnione funkcje w systemie mikroprocesorowym wyróżniamy **pamięć programu** i **pamięć danych** (Tabela 3.1). Pamięć programu to pamięć półprzewodnikowa przeznaczona do przechowywania kodu programu realizowanego przez procesor, natomiast pamięć danych służy do przechowywania danych i wyników obliczeń w czasie realizacji programu przez procesor. Ze względu na własności użytkowe układy pamięci dzielimy na **pamięci zapisywalno-odczytywalne** i pamięci przeznaczone **tylko do odczytu**, tzw. pamięci stałe.

Pamięci zapisywalno-odczytywalne [40] znane są powszechnie pod nazwą **RAM** - od słów (ang. *Random Access Memory*), czyli pamięć o swobodnym dostępie. Istnieją dwa rodzaje pamięci - **dynamiczna DRAM** (ang. *Dynamic RAM*) i **statyczna SRAM** (ang. *Static RAM*).

Pamięci stałe służą do przechowywania programów i wartości stałych. Umieszczenie oprogramowania w pamięci stałej zabezpiecza je przed świadomymi i przypadkowymi modyfikacjami. Zatem jedną z najistotniejszych cech tych pamięci jest ich **nieulotność**, co oznacza, że zawartość pamięci nie ulega zmianie po zaniku napięcia zasilania. Ze względu na sposób wykonania pamięci stałe dzielą się na: ROM (ang. *Read Only Memory*), EPROM (ang. *Erasable Programmable ROM*), FLASH (ang. *Bulk Erasable Non-Volatile Memory*).

Tabela 3.1 Podstawowe układy pamięci półprzewodnikowej

Przeznaczenie pamięci	Konieczne własności	Stosowane typy układów scalonych
Pamięć programu	Zawartość pamięci nie może zaniknąć wraz z wyłączeniem napięcia zasilania	• Pamięci programu bez możliwości zmiany ich zawartości przez użytkownika (**ROM**); • Pamięci programu z możliwością kasowania dotychczasowej zawartości promieniami ultrafioleto-

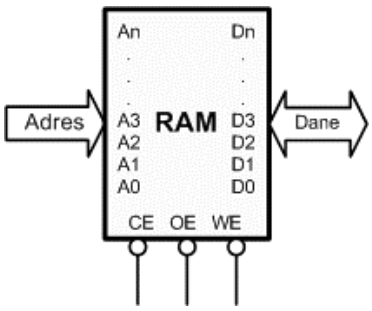
		wymi i wprowadzania nowej zawartości przy pomocy zewnętrznego programatora (**EPROM**); • Pamięci programu kasowane elektryczne z możliwością kasowania dotychczasowej zawartości i wprowadzania nowej bezpośrednio w systemie mikroprocesorowym (**FlashEPROM**).
Pamięć danych	Podczas pracy w systemie mikroprocesorowym musi istnieć możliwość zarówno czytania jak i wpisywania do pamięci	• Pamięci danych statyczne o krótkich czasach dostępu, prostsze w obsłudze przez jednostkę centralną ale droższe (SRAM); • Pamięci danych dynamiczne, pamięci tańsze ale ich obsługa przez jednostkę centralną jest bardziej skomplikowana. Polega to na konieczności wykonywania w krótkich odstępach czasu określonych operacji na pamięci (tzw. odświeżaniu). W przeciwnym przypadku dane zawarte w pamięci dynamicznej zanikają (DRAM).

Organizacja pamięci w systemie mikroprocesorowym

Układy pamięci muszą być stosowane w systemach zawierających mikroprocesory. Z definicji mikroprocesora wynika bowiem, że w jego strukturze nie ma pamięci i w systemie musi być ona zastosowana jako zewnętrzny układ scalony. Ale także w systemach zbudowanych na mikrokontrolerach z reguły stosuje się zewnętrzne układy pamięci. Teoretycznie zarówno pamięć jak i urządzenia peryferyjne są częścią składową struktury mikrokontrolera. Ale w praktyce w większości odmian mikrokontrolerów pamięć programu w strukturze mikrokontrolera w ogóle nie jest implementowana lub jest jej niewiele dla danej aplikacji i wtedy musi być ona układem zewnętrznym.

Układy pamięci kontaktują się z otoczeniem poprzez szynę danych, szynę adresowa i szynę sterującą. Szerokość szyny adresowej określa liczbę bitów zapamiętywanych do pamięci lub czytanych z pamięci w trakcie pojedynczej operacji komunikacji z pamięcią. Szyny danych obecnie produkowanych układów pamięci mają szerokość 8- lub 16-bitów. Szerokość szyny adresowej określa maksymalną liczbę komórek, które mogą być zapamiętane w danym układzie pamięci. Liczba bitów pojedynczej komórki jest oczywiście równa szerokości szyny danych. Tę maksymalną liczbę czyli tzw. pojemność pamięci łatwo można obliczyć poprzez podniesienie liczby 2 do potęgi równej szerokości szyny adresowej. Np. stosowana najczęściej w mikrokontrolerach 8-bitowych szyna adresowa o szerokości 16 bitów pozwala przyłączyć do mikrokontrolera maksymalnie 64 kilobajty tj. 65536 8-bitowych komórek pamięci. Sygnały wchodzące w skład szyny sterującej są uzależnione od konkretnego typu pamięci. Często są to trzy sygnały (rysunek 3.2) oznaczane jako:

- CE – sygnał selekcji układu (ang. *Chip Enable*);
- OE – zezwolenie na wyprowadzanie (odczyt) informacji (ang. *Output Enable*);
- WE – zezwolenie na zapis (ang. *Write Enable*).



Rysunek 3.2. Schemat modułu pamięci

Bez sygnału CE nie jest możliwa jakakolwiek operacja na układzie pamięci. Stosowanie tego sygnału w układach pamięci jest niezbędne ponieważ w danym systemie mikroprocesorowym jest stosowane przeważnie kilka układów pamięci. Dzięki odpowiednim operacjom logicznym na poszczególnych liniach szyny adresowej uzyskuje się układ, w którym w danym zakresie przestrzeni adresowej aktywna jest tylko pojedynczy wybrany układ pamięci. Jest to niezbędne dla poprawnego działania systemu mikroprocesorowego.

Sygnał OE używany jest podczas operacji czytania z pamięci. Po jego doprowadzeniu na szynie danych układu pamięci pojawia się zawartość tej komórki pamięci, na którą wskazywał stan szyny adresowej bezpośrednio przed doprowadzeniem sygnału OE. Z czytaniem wiąże się ważny parametr układów pamięci, tzw. czas dostępu (ang. *Read Access Time*). Parametr ten – podawany najczęściej w nanosekundach (ns) – określa czas opóźnienia pomiędzy doprowadzeniem do układu pamięci sygnału OE a pojawieniem się na szynie danych zawartości danej komórki pamięci. W najszybszych pamięciach produkowanych obecnie czas ten jest mniejszy niż 10 ns.

Sygnał WR używany jest podczas operacji wpisywania do pamięci. Po jego doprowadzeniu do tej komórki pamięci, na którą wskazywał stan szyny adresowej bezpośrednio przed doprowadzeniem sygnału WR wpisywana jest wartość zgodna z aktualnym stanem szyny danych.

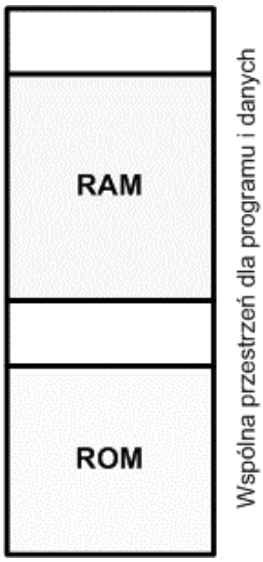
Modele pamięci

Bardzo istotną sprawą związaną z układami pamięci jest model pamięci stosowany w mikrokontrolerach i mikroprocesorach [10].

Jeżeli w systemie mikroprocesorowym jest kilka zewnętrznych układów pamięci to magistrala danych dochodzi do każdego z nich. Jednakże operacja zapisu lub czytania do określonej komórki pamięci dotyczy jednego konkretnego układu. Zatem w danym momencie powinien być aktywny tylko ten jeden układ pamięci. Problem modelu pamięci sprowadza się do odpowiedzi na pytanie w jaki sposób to osiągnąć.

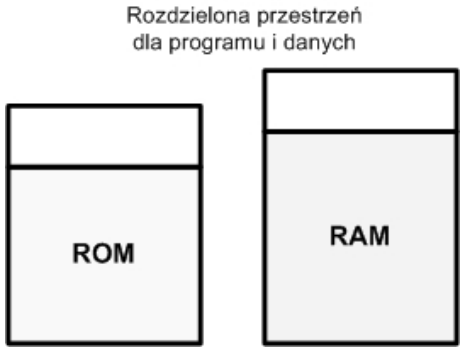
Najbardziej oczywistą metodą rozwiązania tego problemu jest przypisanie każdej komórce pamięci niepowtarzalnego adresu. Warunkiem tego jest dostateczna szerokość magistrali adresowej, tak aby była on w stanie wygenerować wystarczającą dla wszystkich komórek pamięci liczbę adresów. Taki model pamięci nosi nazwę modelu liniowego lub modelu von Neumann'a (rysunek 3.3). Taki model pamięci stosowany jest obecnie najczęściej. Używają go prawie wszystkie współczesne mikroprocesory oraz większość mikrokontrolerów 16- i 32-bitowych.

Odmienne rozwiązanie stosuje się w większości mikrokontrolerów 8-bitowych, np. w rozpowszechnionej rodzinie 8051. Stosuje się tu umieszczanie układów zawierających pamięć programu i układów zawierających pamięć danych w osobnych przestrzeniach adresowych. Taki model pamięci nosi nazwę modelu typu Harvard (rysunek 3.4). W modelu tym dwie komórki pamięci mogą mieć jednakowe adresy pod warunkiem, że leżą one w innych przestrzeniach adresowych.



Rysunek 3.3. Model pamięci typu von Neumann'a

Rozwiązanie taki tylko pozornie zmniejsza wymaganą wielkość przestrzeni adresowej a zatem liczbę wyprowadzeń magistrali adresowej. Zmniejszenie liczby wyprowadzeń magistrali adresowej odbywa się kosztem zwiększenia wyprowadzeń szyny sterującej. Dodatkowe sygnały sterujące muszą być generowane przez jednostkę centralną w celu wskazania przestrzeni adresowej, w której dokonuje się operacji odczytu lub zapisu komórki pamięci o danym adresie.



Rysunek 3.4. Model pamięci typu von Harvard

Każdy z wymienionych modeli pamięci ma swoje wady i zalety. Obydwa modele są wykorzystywane we współczesnych systemach mikroprocesorowych.

Pamięć w systemach mikroprocesorowych bazujących na mikroprocesorach składa się niezależnych zewnętrznych układów scalonych. Podłączenie tych pamięci do mikroprocesora jest od strony koncepcyjnej jasne. Odbywa się ono za pomocą magistrali danych, adresowej i sterującej. Są one dostępne zarówno od strony mikroprocesora jak i układów pamięci.

Natomiast w systemach mikroprocesorowych bazujących na mikrokontrolerach dołączenie zewnętrznych układów pamięci jest możliwe jedynie ze pośrednictwem wbudowanych układów peryferyjnych, ponieważ jedną z możliwości wykorzystania urządzeń peryferyjnych mikrokontrolerów jest wyprowadzenie przez nie szyny adresowej, danych oraz sterującej.

3.3. Układy wejścia-wyjścia i podstawowe układy peryferyjne

Podobnie jak w przypadku układów pamięci układy wejścia-wyjścia stosowane są zarówno w systemach zawierających mikroprocesory jak i mikrokontrolery [31, 33, 36].

W przypadku mikroprocesorów jest to oczywiste, ponieważ nie ma tych układów w strukturze układu scalonego mikroprocesora a są one niezbędne min. do komunikacji systemu z użytkownikiem układy np. układy umożliwiające w podłączeniu do systemu klawiatury.

Natomiast stosowanie zewnętrznych układów wejścia-wyjścia w systemach zawierających mikrokontrolery jest niezbędne tylko wówczas gdy projektant systemu wśród setek odmian mikrokontrolerów oferowanych na rynku nie znalazł odpowiadającego mu układu zawierającego w strukturze potrzebne w danym zastosowaniu wbudowane w strukturę mikrokontrolera układy peryferyjne.

Różnorodność układów wejścia-wyjścia jest tak duża, że trudno nawet dokonać ich klasyfikacji. Za podstawowe układy peryferyjne można uznać te, które zostały umieszczone w strukturze pierwszych mikrokontrolerów (np. 8051) [12, 38]

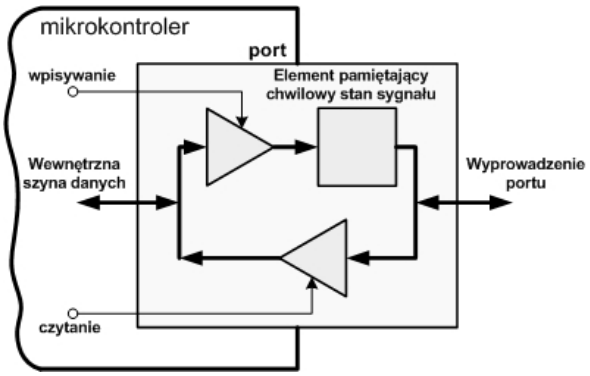
i umieszczane są zawsze w strukturach mikrokontrolerów produkowanych obecnie. Są to:

- układy wejść-wyjść cyfrowych - porty (ang. *ports*);
- układy czasowe (ang. *Timers / Counters Unit*);
- sterowniki komunikacji szeregowej (ang. *Serial Communication Unit*);
- układy specjalizowane (przetworniki analogowo-cyfrowe, układy do generowania sygnałów cyfrowych, np. wyjścia typu PWM, układy nadzorujące).

Współczesne systemy mikroprocesorowe zawierać mogą cały szereg wymienionych wyżej układów wejścia-wyjścia i peryferyjnych. Ich wybór jest specyficzny dla danej aplikacji. Poniżej zostaną omówione jedynie podstawowe układy peryferyjne zawarte w strukturze mikrokontrolerów. Każdy projektant bądź programista systemu mikroprocesorowego opartego na mikrokontrolerze musi się tymi urządzeniami spotkać.

Porty

Do równoległej wymiany danych cyfrowych służą zespoły linii wejścia-wyjścia zwane portami [10, 12, 19, 34, 38]. W zdecydowanej większości rozwiązań porty zawierają po 8 linii. Liczba portów i ich wyprowadzeń (pinów) jest ograniczona przez liczbę wyprowadzeń obudowy mikrokontrolera. Porty są podstawowymi i najważniejszymi układami peryferyjnymi. Wszystkie pozostałe urządzenia peryferyjne wykorzystują funkcje portów.



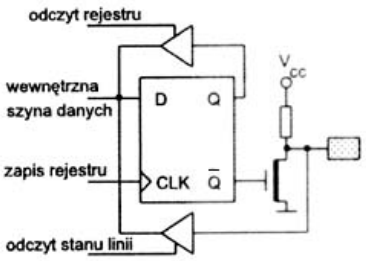
Rysunek 3.5. Budowa portu

Zadaniem portów (rysunek 3.5.) jest doprowadzanie sygnałów binarnych z otoczenia mikrokontrolera do jego wnętrza oraz odwrotnie – wyprowadzanie sygnałów z wnętrza mikrokontrolera do jego otoczenia. Za transport informacji wewnątrz mikrokontrolera odpowiedzialna jest szyna danych. Zatem czytanie danych przez port polega na doprowadzaniu chwilowych stanów wyprowadzeń portu (tzn. istniejących w momencie operacji czytania) do wewnętrznej szyny danych układu. Natomiast operacja wpisania do portu powoduje, że chwilowy stan wewnętrznej szyny danych zostaje zapamiętany na wyprowadzeniach portu. Stan wyprowadzeń portu pozostaje niezmienny póki nie nastąpi kolejna operacja wpisywania do portu. Wyróżnia się następujące typy portów:

- porty dwukierunkowe;
- porty jednokierunkowe wejściowe;
- porty jednokierunkowe wyjściowe.

Porty - układy wejść-wyjść cyfrowych 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, przekaźniki, silniki, czy nawet inne mikrokontrolery.

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 Rysunku 3.6. 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 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.



Rysunek 3.6. Budowa linii wejść-wyjść cyfrowych

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.

Układy czasowe

Współpraca systemu mikroprocesorowego z otoczeniem w czasie rzeczywistym wymaga odliczania czasu lub generowania złożonych sekwencji binarnych. Jest to realizowane przez specjalizowane bloki nazywane licznikami (ang. *counters*) lub układami czasowymi (ang. *timers*) [7, 18, 34] z dokładnością oscylatora kwarcowego. Liczba liczników i ich długość wyrażona w bitach, różnią się dla konkretnych typów mikrokontrolerów. Większość posiada przynajmniej dwa liczniki 16-bitowe, natomiast każdy mikrokontroler posiada choćby jeden licznik 8-bitowy.

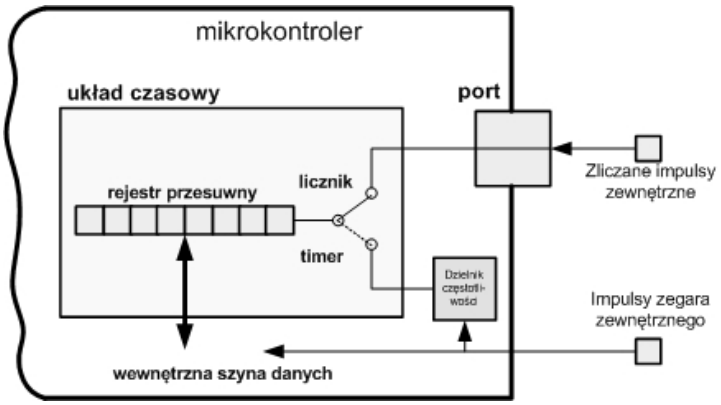
Najprostszą strukturę takiego układu pokazano na Rysunku 3.7. Składa się on z rejestru o określonej długości i rejestru zawierającego np. bity przepełnienia, ustawiające tryb pracy, bit startu zliczania licznika, itd. Rejestr przesuwny zliczający liczbę zmian poziomów doprowadzonych do niego sygnałów jest dostępny z poziomu programu użytkownika, jako rejestr (dla liczników 8-bitowych) lub para rejestrów (dla liczników 16-bitowych) o określonych adresach w obszarze pamięci danych.

Podstawowe układy czasowe mogą one pracować w dwóch konfiguracjach:

- jako właściwe tajmery (ang. *Timers*) - są one wtedy taktowane sygnałem zegarowym z wewnętrznego zegara układu

i zazwyczaj wykorzystywane w programie użytkownika jako wzorce czasu;

- jako liczniki (ang. *Counters*) - są one wtedy taktowane zewnętrznymi sygnałami doprowadzanymi poprzez linie wejściowe portów i są zazwyczaj wykorzystywane w programie użytkownika jako liczniki zmian poziomów sygnałów zewnętrznych.



Rysunek 3.7. Budowa układu czasowego

Ważną właściwością liczników jest sposób zliczania impulsów. Liczniki mogą zliczać impulsy w górę, tzn. inkrementować lub w dół czyli dekrementować.

W praktyce mikrokontrolery są wyposażone w bardziej rozbudowane układy licznikowe. Liczniki te są używane zazwyczaj do realizacji następujących funkcji:

- określenia (mierzenia) odstępów czasu między zdarzeniami zachodzącymi na zewnątrz, sygnalizowanymi przez impulsy elektryczne doprowadzone do mikroprocesora. Funkcja ta bywa nazywana rejestracją zdarzeń (ang. *input event capture*),

- generowania impulsów (sekwencji impulsów) w odstępach czasu o zaprogramowanej wartości lub przebiegu okresowego o zadanej częstotliwości,

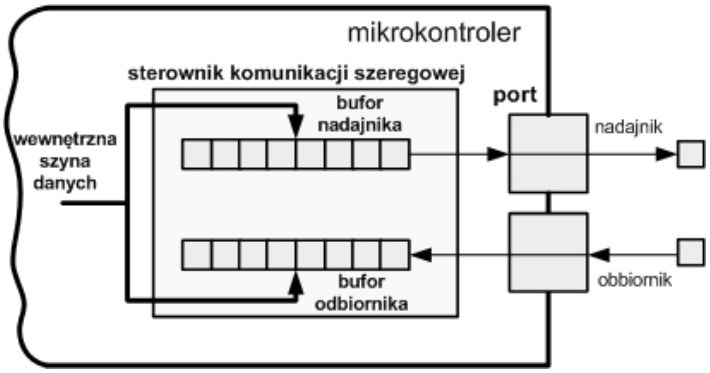
- generowania sygnałów impulsowych o określonym czasie trwania lub sygnałów okresowych o zadanym współczyn-

niku wypełnienia, nazywanych sygnałami PWM (ang. *puls width modulation*),

- sterowania szybkością transmisji w portach szeregowych, zarówno w trybie synchronicznym, jak i asynchronicznym,
- realizacji zadań licznika nadzorcy – watchdoga.

3.4. Interfejsy i sterowniki do szeregowej transmisji danych

Sterowniki komunikacji szeregowej służą do wymiany informacji pomiędzy systemem mikroprocesorowym, a jego otoczeniem. Przesyłanie danych odbywa się w sposób szeregowy [26, 30, 42]. Zasadę działania tych układów pokazano na Rysunku 3.8.



Rysunek 3.8. Budowa sterownika komunikacji szeregowej

Urządzenie to umożliwia wysłanie zawartości określonych rejestrów (dostępnych z poziomu programu użytkownika) w postaci szeregowej poprzez określone wyprowadzenia portu. Oznacza to, że na wyjściu linii portu pojawia się ciąg binarny odpowiadający zawartości wysyłanego rejestru. Jest to funkcja nadajnika (ang. *Transmiter*). W funkcji odbiornika (ang. *Reciver*) sterownik komunikacji szeregowej potrafi przetworzyć doprowadzony na wejście określonej linii portu ciąg binarny na zawartość rejestru dostępnego z poziomu programu użytkownika.

Wyróżnia się dwa rodzaje transmisji szeregowej:

- asynchroniczną;
- synchroniczną.

Dane przesyłane **asynchronicznie nie są związane z żadnym sygnałem synchronizującym**, w szczególności nie towarzyszy im sygnał zegara. Transmisja przebiega zwykle bajtami przesyłanymi w postaci ciągów bitów. Oprócz ośmiu bitów danych przesyła się dodatkowo bit startu oraz opcjonalnie bit parzystości do detekcji błędów i bit stopu. Pomiędzy nadajnikiem, a odbiornikami **musi być ustalona częstotliwość przesyłania danych**.

Przy **transmisji synchronicznej** równolegle z ciągiem bitów danych przesyła się **sygnał synchronizujący** (zegarowy), który określa chwile, w których stan linii danych odpowiada ważnym wartościom kolejnych bitów. Po każdym bajcie może być dodatkowo przesłany bit parzystości.

Sterownik komunikacji szeregowej (w skrócie: **port szeregowy**) nazywa się **dwukierunkowym** (dupleksowym), jeśli może równocześnie odbierać i nadawać dane. Jest to możliwe, gdy jest wyposażony w dwie oddzielne linie do nadawania i do odbioru.

Można wyróżnić między innymi następujące interfejsy szeregowe [8, 30] stosowane w mikrokontrolerach i systemach mikroprocesorowych:

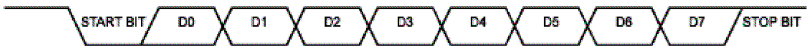
- UART (ang. *Universal Asynchronous Receiver/Transmitter*),
- SPI (ang. *Serial Peripherial Interface*),
- 1-Wire (ang. *One Wire*),
- I2C (ang. *Inter-Integrated Circuit*),
- CAN (ang. *Cotroller Area Network*),
- USB (ang. *Universal Serial Bus*).

Interfejs UART

Interfejs ten przewidziany jest zazwyczaj do zapewnienia łączności systemu mikroprocesorowego z komputerem PC [6, 48] lub innym mikrokontrolerem. Mikrokontrolery często posiadają rozbudowane układy UART posiadające własne generatory sygnałów odniesienia, bufory, dekodery kodów sterujących, mogące pracować w trybie dwukierunkowym

z protokołem potwierdzeń sprzętowych (Xon/Xoff). Prostsze mikrokontrolery mają jeden układ UART, natomiast bardziej rozbudowane mają ich kilka.

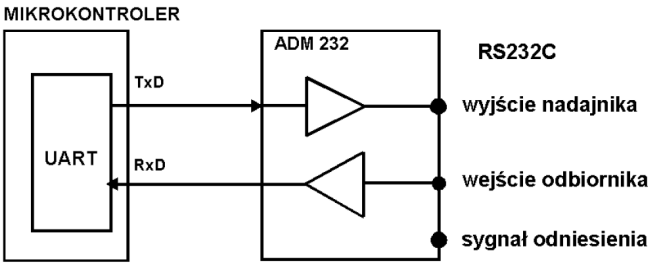
Port szeregowy UART kontaktuje się ze światem zewnętrznym za pomocą dwóch wyprowadzeń: **wejścia odbiornika** (oznaczanego najczęściej RxD) i **wyjścia odbiornika** (TxD). Interfejs UART jest interfejsem **asynchronicznym**, najczęściej posiada swój własny **generator taktujący**.



Rysunek 3.9. Format danych dla standardu UART

Format danych w tym standardzie pokazano na Rysunku 3.9. Jak widać, transmisja zaczyna się od **bitu startu**, po którym następuje **osiem bitów danej** (czasami dziewięć, gdzie dziewiąty bit jest najczęściej bitem parzystości) i jednego **bitu stopu** (w skrócie: 8N1).

Jak wspomniano, interfejs UART przeważnie służy do komunikacji systemu mikroprocesorowego z komputerem PC. Odbywa się to najczęściej za pomocą interfejsu **standardu RS232** [30]. Sposób realizacji sprzętowej standardu RS232C pokazano na Rysunku 3.10.



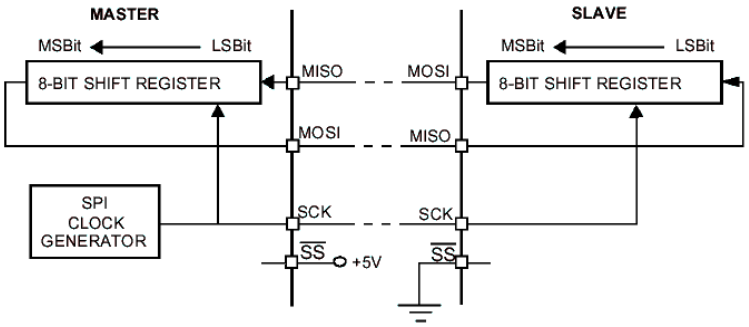
Rysunek 3.9. Realizacja sprzętowa standardu RS232C

Zadaniem układu ADM232 jest konwersja sygnałów pomiędzy poziomem TTL na wyprowadzeniach mikrokontrolera a poziomem sygnału +/-12V wykorzystywanym podczas transmisji łączem RS232. Łącze RS232C wymaga co najmniej 3 przewodów: linii nadajnika, odbiornika oraz linii odniesienia, czyli tzw. Masy sygnałowej. Jest to zatem łącze asymetryczne. Łącze RS232 umożliwia komunikację ze sobą tylko dwóch jednostek przy zastosowaniu transmisji typu „full duplex". Oznacza to, że danym momencie możliwa jest na linii transmisja danych jednocześnie w obydwu kierunkach. Jest to oczywiście wtedy możliwe, jeżeli pozwala na to sterownik komunikacji szeregowej. Układ UART ma

tą możliwość. Łącze typu RS232C stosowane jest powszechnie w komputerach osobistych i dostępne pod oznaczeniami COM (COM1, COM2 itp.).

Interfejs SPI

Interfejs SPI [8] służy do **dwukierunkowej** (ang. *full-duplex*), **synchronicznej**, szeregowej transmisji danych pomiędzy systemem mikroprocesorowym, a zewnętrznymi układami peryferyjnymi, np. przetwornikami A/C i C/A, szeregowymi pamięciami EEPROM, innymi mikrokontrolerami, układami scalonymi z regulowanymi cyfrowo potencjometrami, dekadami pojemnościowymi, itd. Jest interfejsem **trójprzewodowym**: składa się z **dwóch linii synchronicznie przesyłających dane** w przeciwnych kierunkach i **linii z sygnałem zegarowym** synchronizującym ten transfer.



Rysunek 3.11. Przykład połączenia interfejsu ISP pomiędzy układem master i slave

Na rysunku 3.11 pokazano sposób połączenia interfejsu SPI pomiędzy **układem nadrzędnym** (ang. *master*) i **układem podrzędnym** (ang. *slave*), gdzie linia MISO jest wejściem danych, linia MOSI wyjściem danych, a linia SCK wyjściem zegara dla układu master i wejściem zegara dla układu slave. Z rysunku wynika, że protokół transmisji (w najprostszej postaci) wymaga dwóch rejestrów przesuwnych (ang. *shift register*) połączonych w **licznik pierścieniowy** (wejście jest połączone z wyjściem). Transmisja jest synchroniczna - jeden z układów dostarcza sygnału zegara, odseparowanego od sygnału danych, zgodnie ze zboczami zegara taktującego. **Układ generujący sygnał zegara jest określony jako nadrzędny**, bez względu na to czy dane są przez niego nadawane czy odbierane. Wszystkie pozostałe układy na magistrali są określone jako **podrzędne**. Sygnały danych i zegara są przesyłane oddzielnymi jednokierunkowymi liniami. Sygnał zegara nie jest ciągły, nadawany jest jedynie w czasie trwania transmisji. Umożliwia to odbiór poszczegól-

nych bitów danych bez konieczności testowania bitów startu i stopu, charakterystycznego dla transmisji asynchronicznej.

Kiedy układ nadrzędny nadaje dane na linii MOSI do układu podrzędnego, to układ podrzędny odpowiada, wysyłając do układu nadrzędnego dane na linii MISO. Implikuje to dwukierunkową transmisję z jednoczesnym wysyłaniem i odbieraniem danych, synchronizowanym tym samym sygnałem zegarowym, przez oba układy. Zatem bajt transmitowany jest od razu zastępowany bajtem odbieranym. Dzięki temu nie ma „pustych" transmisji: raz aby wysłać bajt, drugi aby go odebrać. Do wysłania bajtu i jego odebrania wystarczy osiem impulsów zegarowych na linii SCK.

W interfejsie tym dane są wpisywane (odbierane) do rejestru szeregowego jednym zboczem zegarowym, a przesuwane (wyprowadzane na zewnątrz – nadawane) drugim. Zwiększa to czas podtrzymania danych odbiornika do ½TCLK – td, gdzie TCLK - okres zegara, td - opóźnienie magistrali. Zatem wypełnienie sygnału zegarowego wynosi około ½. Najczęściej szybkość transmisji nie przekracza 1÷2 Mbit/s.

Podsumowując, właściwości układu SPI z punktu widzenia interfejsu mogą być opisane przy pomocy 3 funkcji interfejsowych:

- **Talker** (nadajnik) - wprowadzenie danej do rejestru szeregowego, przesuwanie i wysyłanie danej z rejestru szeregowego,
- **Listener** (odbiornik) - wczytanie danej odbieranej z rejestru szeregowego po zebraniu całego słowa,
- **Repeater** (pośredniczenie w transmisji) - dane wejściowe są wpisywane do rejestru szeregowego z linii wejściowej, przesuwane i od razu wysyłane na linię wyjściową.

Maksymalna konfiguracja zawiera z reguły wszystkie trzy wymienione funkcje, ale w praktyce spotyka się najczęściej dwie pierwsze.

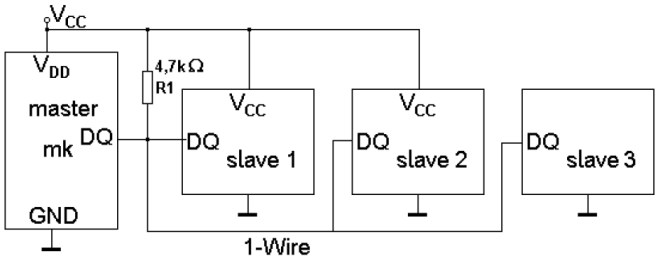
Interfejs 1-Wire

Interfejs **1-Wire** [8] został opracowany przez firmę Dallas Semiconductor (obecnie Maxim) jest on przeznaczony do przesyłania informacji pomiędzy układem nadrzędnym **master** (mk) i układami podrzędnymi **slave** (np. termometry, układy identyfikacji, pamięci SRAM i EEPROM, programowalne klucze). Transmisja odbywa się w obu kierunkach z wykorzystaniem jednego przewodu sygnałowego (oprócz masy), który jed-

nocześnie może być wykorzystany do zasilania układów do niego podłączonych (rysunek 3.12). Dane przesyłane są z prędkością od bliskiej 0 do 16,3 kbps w trybie standard oraz do 115 kbps w trybie overdrive.

Każde z urządzeń podłączonych do magistrali musi mieć wyjście typu otwarty dren, a linia sygnałowa DQ jest połączona do zasilania przez rezystor podciągający o wartości około 5kΩ.

Zatem w stanie bezczynności linia DQ jest w stanie wysokim. Magistrala nie ma ustalonego formatu danych. Sposób przesyłania informacji zależy od konfiguracji układów podrzędnych. Zawsze jednak jako pierwszy jest przesyłany bit najmniej znaczący.



Rysunek 3.12. Schemat połączeń interfejsem 1-Wire

Protokół wymiany danych poprzez magistralę 1-Wire składa się z czterech podstawowych sekwencji:

- inicjalizacji (zerowanie magistrali),
- wysłanie (zapis) zera,
- wysłanie (zapis) jedynki,
- odczyt bita.

Sekwencja inicjalizacji rozpoczyna się wysłaniem na magistralę przez układ master impulsu zerującego o czasie trwania 480÷960µs. Po tym czasie układ master przechodzi w stan odbioru i na magistrali pojawia się stan wysoki. Po zidentyfikowaniu końca impulsu zerującego układ slave odczekuje 15÷60µs i wystawia na magistralę impuls obecności (ang. *presence pulse*) o czasie trwania 60÷240µs. Sekwencja inicjalizacji umożliwia układowi master wykrycie podłączonych do niej układów slave. Po czasie koniecznym do pełnej inicjalizacji układów podrzędnych możliwe staje się przeprowadzenie normalnej transmisji danych.

Transmisja danych do urządzenia podrzędnego polega na generowaniu serii impulsów o poziomie niskim o odpowiedniej długości, która definiuje stany logiczne „0" lub „1". Długość sekwencji zapisu lub odczytu musi się zmieścić w szczelinie czasowej (ang. *time slot*), która wynosi 60÷120µs i jest inicjowana wymuszeniem przez układ master stanu niskiego na magistrali.

Nadanie logicznego „**0**" polega na wygenerowaniu impulsu o czasie trwania 60÷120µs, a następnie na zwolnieniu magistrali i odczekaniu minimum 1µs przed nadaniem następnego bita. Dla logicznej „**1**" generowany impuls trwa 1÷15µs, natomiast wymagany czas bezczynności wynosi 60µs.

Odczyt wartości bita przesyłanego przez układ podrzędny polega na generacji przez układ master impulsu o czasie trwania minimum 1µs (zazwyczaj stosuje się impulsy 3÷5µs), a następnie na zwolnieniu linii DQ i sprawdzeniu jej stanu logicznego przed upływem 15µs od rozpoczęcia sekwencji odczytu.

W większości przypadków interfejs 1-Wire implementuje się w systemie mikroprocesorowym w sposób programowy. Jednak, ze względu na rosnącą popularność interfejsu wynikającą z faktu, iż składa się on tylko z jednej linii sygnałowej, która może zasilać urządzenia podrzędne (o niewielkiej mocy pobieranej) pojawiają się mikrokontrolery z kontrolerami tego interfejsu (np. DS80C400 firmy Dallas Semiconductor).

Interfejs I2C

Standard ten został opracowany przez firmę Philips [8]. Jest stosowany w urządzeniach powszechnego użytku, zwłaszcza audio-video, telekomunikacji i systemach elektroniki przemysłowej. Transmisja danych odbywa się **szeregowo**, w **dwóch kierunkach**, przy użyciu **dwóch linii**. Jedną z nich **SCL** (ang. *serial clock line*) przesyła się **impulsy zegarowe** synchronizujące transmisję, natomiast drugą **SDA** (ang. *serial data line*) transmituje się w dwóch kierunkach **dane**.

W transmisji interfejsem I2C uczestniczy układ **nadrzędny** (ang. *master*) oraz jeden lub więcej układów **podrzędnych** (ang. *slave*). Transmisja bloku danych jest inicjowana wówczas, gdy stan linii SDA zmieni się z wysokiego na niski, podczas gdy linia zegarowa SCL znajduje się w stanie wysokim. Sytuacja taka jest nazywana **warunkiem startowym (Start)**. Koniec transmisji ma miejsce wtedy, gdy linia SDA zmieni stan z niskiego na wysoki, podczas gdy linia zegara jest w stanie wysokim. Jest to warunek nazywany **Stop**. Stan linii danych może być

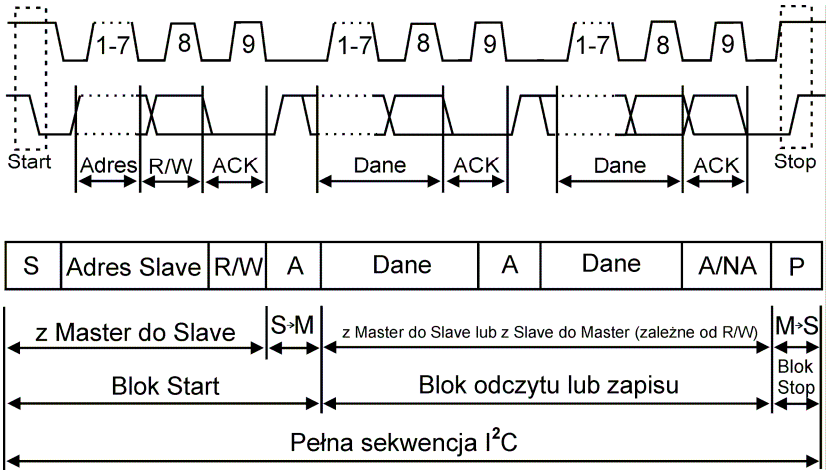
zmieniany wówczas, gdy linia zegarowa znajduje się w stanie niskim. Od chwili wystąpienia warunku Start do chwili odległej o określony odstęp czasu od wystąpienia warunku Stop szyna jest zajęta.

Informacje są przesyłane bajtami, w postaci szeregowej, przy czym jako pierwszy wysyłany jest najbardziej znaczący bit. Po przesłaniu danych wysyłany jest **sygnał potwierdzenia** oznaczony symbolem A lub ACK (ang. *acknowledge*). Dziewiąty impuls zegara towarzyszący potwierdzeniu jest generowany przez układ master. Na ten sygnał urządzenie nadające musi zwolnić linię SDA, wprowadzając swoje wyjście SDA w stan wysoki lub w stan wysokiej impedancji, natomiast odbiornik potwierdza odbiór sygnałem SDA=0. Liczba bajtów przesłanych w jednej **sesji, to jest między stanem Start i Stop**, jest nieograniczona.

Wysłanie bitu potwierdzenia przez odbiornik po odebraniu każdego bajtu danych jest obligatoryjne. Jeśli potwierdzenie nie zostaje wysłane, linia SDA musi znaleźć się w stanie wysokim lub w stanie wysokiej impedancji, by umożliwić układowi master wygenerowanie warunku Stop i zakończenie transmisji. Brak potwierdzenia może nastąpić na przykład wtedy, gdy odbiornik pracuje w czasie rzeczywistym i jest zajęty obsługa jakiegoś pilnego zadania.

Jeśli odbiornikiem jest układ master, to po odebraniu ostatniego bajtu w sekwencji nie wysyła sygnału potwierdzenia NA (ang. *not acknowledge*). Układ slave musi wtedy zwolnić linię SDA, by umożliwić układowi master wygenerowanie warunku Stop kończącego transmisję.

Na rysunku 3.13 przedstawiono kompletny cykl transmisji w formacie standardu I2C. Po wygenerowaniu warunku Start układ master wysyła na szynę **adres układu slave**, z którym chce współpracować. **Adres składa się z siedmiu bitów**, po którym następuje **ósmy bit R/W** określający **kierunek transmisji**. Jeśli R/W=0, dane będą przesyłane z układu master do układu slave, natomiast gdy R/W=1, kierunek transmisji będzie odwrotny. Początkowa sekwencja bitów przesyłanych w trakcie każdej sesji, złożona z bitu Startu, adresu urządzenia slave i bitu kierunku jest nazywana **blokiem Start**. Dalsza część transmisji polega na przesyłaniu kolejnych bajtów danych, zgodnie z protokołem pokazanym na rysunku. Ostatnią fazą sesji jest wysłanie przez układ master warunku Stop.

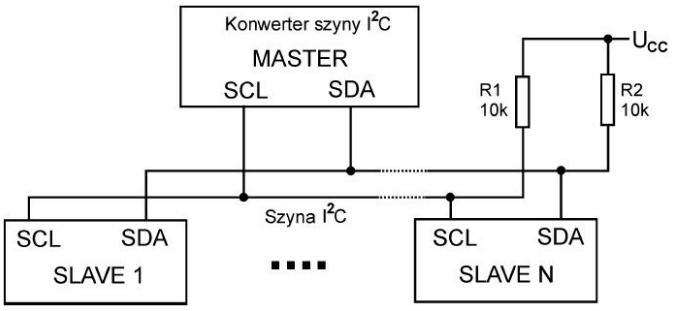


Rysunek 3.13. Protokół transmisji szeregowej standardu I2C

Pełna sesja składa się zatem **z bloku Start, przynajmniej jednego cyklu przesłania bajtu danych i bitu Stop**. Jeśli układ master zamierza w jednej sesji wysyłać informacje w kilku blokach do tego samego lub do różnych urządzeń, może nie kończyć sesji warunkiem Stop, lecz po przesłaniu bloku danych powtórzyć warunek Start z tym samym lub nowym adresem urządzenia slave.

Przemysłowa norma magistrali I2C zakłada możliwość taktowania transmisji sygnałem zegarowym SCL o częstotliwości od **0 do 100kHz**. Wymaga się przy tym, by stan niski impulsów zegarowych trwał przynajmniej 4,7µs, natomiast stan wysoki był nie krótszy niż 4µs. W razie potrzeby układ slave może spowolnić transmisję, utrzymując linię zegara SCL w stanie niskim po odebraniu bajtu danych od mk. Stan taki jest nazywany **stanem oczekiwania** i oznaczany symbolem **WAIT**. Realizacja stanów oczekiwania na magistrali wymaga, by układ master przed wysłaniem kolejnego bajtu przełączył linię SCL w stan wysoki lub w stan wysokiej impedancji, po czym odczytał stan na SCL.

Układy współpracujące z magistralą I2C (rysunek 3.14) muszą być wyposażone w wyjścia z otwartym drenem dla linii SCL i SDA. Każda z tych linii jest połączona ze źródłem napięcia zasilającego UCC przez rezystor 10kΩ.



Rysunek 3.14. Połączenie urządzeń z magistralą I2C

Do jednego układu master można przyłączyć dowolną liczbę układów slave, jednak pod warunkiem, że pojemność połączeń nie przekroczy maksymalnej wartości równej 400pF.

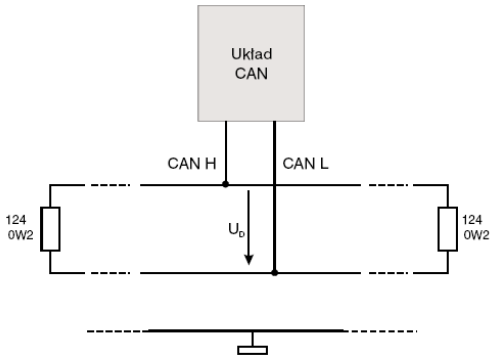
Interfejs CAN

Interfejs CAN [14, 15] stosuje się głównie w motoryzacji, w lotnictwie, a także w sterownikach przemysłowych. Założenia projektowe CAN koncentrują się na trzech właściwościach systemu transmisyjnego:

- **szybkości przesyłania danych** łączem szeregowym;
- **odporności transmisji** na zakłócenia elektromagnetyczne;
- możliwości **dzielenia wspólnych danych** przez rozproszone systemy mikroprocesorowe realizujące cząstkowe funkcje sterowania.

Szyna CAN jest **asynchroniczną** szyną transmisji szeregowej z tylko jedną **linią transmisyjną**. Ma strukturę **otwartą**, tzn. może być rozszerzana o nowe **węzły** (odbiorniki/nadajniki), oraz **liniową**, czyli nie zawiera pętli. Minimalna konfiguracja CAN składa się z dwóch węzłów. Liczba węzłów w trakcie pracy szyny może się zmieniać bez wpływu na działanie szyny. Właściwość ta oznacza możliwość dynamicznego konfigurowania węzłów, na przykład wyłączenia węzłów, które w wyniku działania procedur diagnostycznych zostały uznane za niesprawne.

Poszczególne węzły są przyłączone do szyny na zasadzie funkcji „**zwarty iloczyn**” (ang. *wired-AND*). Przy braku sterowania szyna znajduje się w stanie nazywanym **recesywnym** R (ang. *recessive*), któremu odpowiada stan logiczny „1”. Stan ten może być zmieniony na stan logiczny „0”, jeśli choć jeden węzeł wymusi na szynie poziom **dominujący** D (ang. *dominant*).



Rysunek 3.15. Połączenie urządzenia do magistrali CAN

Najpopularniejszy sposób sprzętowej realizacji szyny jest użycie **pary skręconych przewodów** oznaczonych symbolami **CAN_H** i **CAN_L**. Linie te (rysunek 3.15) przyłącza się do węzłów bezpośrednio lub za pośrednictwem buforów /nadajników/odbiorników. Po obu stronach linii dołącza się rezystory. Maksymalna szybkość transmisji szyną CAN wynosi 1Mbps przy długości linii nie przekraczającej 40m. Im dłuższa linia tym prędkość transmisji mniejsza, np. dla długości przewodów 1km prędkość transmisji wynosi już 40kbps.

Dane przesyłane szyną są transmitowane metodą NRZ (ang. *Non Return to Zero*). Adresy odbiorników (**identyfikatory**) są przesyłane jako integralna część przekazu. Ponieważ interfejs składa się wyłącznie z jednej linii konieczny jest arbitraż szyny. Wykonuje się go metodą CSMA/CD z opcją NDA (ang. *carrier sense multiple access/collision detection with nondestructive arbitration*).

Specyfika standardu CAN przewiduje **11-bitowe identyfikatory**, czyli umożliwia współpracę 2048 węzłów. Najnowsza specyfikacja 2.0B (ang. *extended CAN*) posiada identyfikator 29-bitowy, co daję teoretycznie możliwość adresowania 536 milionów współpracujących ze sobą urządzeń.

Interfejs USB

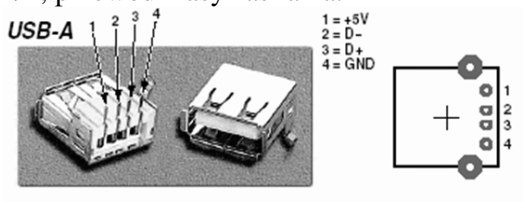
Interfejs ten [6] został opracowany jako **uniwersalny interfejs szeregowy** wbudowany w architekturę komputerów PC. Jego przeznaczeniem jest współpraca komputerów z urządzeniami: przemysłowymi, urządzeniami powszechnego użytku oraz integracja z sieciami telekomunikacyjnymi. Prędkość transferu danych wynosi 1,5 lub 12 Mbit/s dla standardu USB 1.1, natomiast dla standardu USB 2.0 jest ona równa 480 Mbit/s.

Przyjęto następującą topologię systemu z interfejsem USB, jeden **host** (komputer zarządzający) połączony z pierwszym **Hubem**, do którego można podłączyć na zasadzie drzewa kolejne Huby lub urządzenia. Taka topologia zapewnia możliwość współpracy różnych urządzeń poprzez wyspecjalizowane Huby, które z kolei współpracują ze sobą tworząc jednolity system, wygodny do użycia przez końcowego użytkownika dzięki :

- prostemu i znormalizowanemu okablowaniu,
- izolacji elektrycznej,
- samoidentyfikacji urządzeń,
- automatycznej konfiguracji ich sterowników,
- automatycznej rejestracji w systemie,
- niskich kosztów sprzętu i okablowania.

Wszystkie elementy systemu połączone są magistralą (rysunek 3.16) składającą się z czterech linii:

- **Vbus**, przewód zasilania +5 V ;
- **D+**, przewód symetrycznej skrętki sygnałowej;
- **D-**, przewód symetrycznej skrętki sygnałowej;
- **GND**, przewód masy zasilania.



Rysunek 3.16. Złącze i magistrala interfejsu USB

Typowym kablem sygnałowym jest skrętka ekranowana o impedancji charakterystycznej 90Ω ±15[%] i maksymalnej długości 5m. Maksymalne opóźnienie sygnału między punktami końcowymi, tzn. hostem i urządzeniem musi być mniejsze od 70ns. Do transmisji danych interfejsem USB wykorzystuje się kodowanie danych **NRZI** (ang. *Non-Return to Zero Inver on ones*). Kodowanie to polega na braku zmiany poziomu dla jedynki logicznej i zmianie poziomu dla zera logicznego.

Komunikacja między urządzeniami przebiega w następujący sposób: urządzenie fizyczne zostaje zidentyfikowane przez oprogramowanie typu klient, zainstalowane na komputerze (host). Następnie zostaje zainicjowane połączenie pomiędzy urządzeniem a hostem, umożliwiające przesyłanie komunikatów i danych. Dane odebrane z urządzenia zostają sformatowane zgodnie z wymaganiami protokołu USB i przekazane do hosta dzięki współpracy oprogramowania systemu USB, oprogramowania hosta oraz współpracującego sprzętu.

3.5. Urządzenia wejściowe i wyjściowe sterowników maszyn i urządzeń mechatronicznych

Urządzenia wejściowe

Do podstawowych urządzeń wejściowych systemów wbudowanych [7, 20, 25] (sterowników maszyn i rządzeń mechatronicznych) zaliczamy różnorodne elementy manipulacyjne służące zadawaniu stanów poszczególnym urządzeniom. Są one podstawowym wyposażeniem panelu operatora maszyn i pojazdów. Do tej grupy elementów zaliczamy: wyłączniki, przełączniki, przyciski, pokrętła, klawiatury, itp.

Do najpowszechniej używanych manipulatorów zaliczamy przełączniki. Można je znaleźć praktycznie na każdym pulpicie. Służą przede wszystkim do uruchamiania bądź wyłączania wybranych urządzeń i systemów. Przełącznik jest mechanicznym urządzeniem używanym do włączania i wyłączania obwodu elektrycznego. Mechanizm znajdujący się w środku przełącznika zwiera bądź rozwiera ze sobą właściwe wyprowadzenia. Podobnie działają przełączniki obrotowe używane do sterowania wieloma urządzeniami naprzemiennie za pomocą jednego sygnału wejściowego. Przełącznik obrotowy używany jest do przełączania trybów pracy urządzeń. Stosowany jest na przykład do zmiany zakresów pracy wskaźników i monitorów lub wyboru odpowiedniego dla danej fazy pracy maszyny zestawu informacji.

Innymi często stosowanymi elementami manipulacyjnymi są potencjometry i impulsatory. Typowym zastosowaniem potencjometru pełnią-

cego rolę dzielnika napięcia jest regulacja prądu lub napięcia w urządzeniach elektrycznych. Potencjometry są głównie używane do regulacji oświetlenia oraz podświetlenia pulpitów. Natomiast impulsatory nazywane enkoderami obrotowymi zastępują potencjometry w aplikacjach wymagających skokowej zmiany wartości napięcia. Charakterystyczną cechą tych urządzeń jest stała, niezmienna ilość impulsów na wyjściu. Impulsatory używa się do ustawiania wartości wybranego parametru.

Ważną grupę elementów wykorzystywanych do budowy interfejsu sprzętowego stanowią mikroprzyciski zwane mikroswitch`ami lub przyciskami zwiernymi. Elementy te służą do chwilowego zamykania obwodu elektrycznego. Mikroprzyciski stanowią podstawę różnorodnych klawiatur i są stosowane do uruchamiania wybranych trybów pracy poszczególnych urządzeń. Mikroswitch`e można znaleźć na pulpicie układów zarządzania klimatyzacją, sterowania siedzeniami, otwierania drzwi i elektrycznych szyb, obsługi telefonu, centralnego komputera, itp.

Urządzenia wyjściowe

Do grupy podstawowych urządzeń wyjściowych systemów mikroprocesorowych [7, 20, 25] zaliczamy przede wszystkim urządzenia, które służą do sygnalizacji różnorodnych zdarzeń świetlnych, do zobrazowania komunikatów w postaci tekstowej oraz do wyświetlania informacji w zaawansowanej postaci graficznej. Do tych urządzeń zaliczamy żarówki, diody i wyświetlacze siedmiosegmentowe LED oraz ciekłokrystaliczne wyświetlacze tekstowe i graficzne. Nowym trendem w systemach mechatronicznych jest zastępowanie dotychczasowych prostych metod zobrazowania informacji zaawansowanymi wizualizacjami na wskaźnikach graficznych.

Najdłużej stosowanymi elementami sygnalizacyjnymi są żarówki. Stosuje się je do podświetlania tabliczek sygnalizujących wybraną opcję określającą stan poszczególnych urządzeń i podsystemów. We współczesnych kokpitach maszyn i pojazdów, wymienione zadania realizują diody LED, a żarówki są stosowane głównie do podświetlania pulpitów, napisów i znaków wskazujących sytuacje awaryjne. Małe wymiary, budowa odporna na udary, niski pobór mocy to niektóre z zalet, które pozwoliły na zastąpienie żarówek diodami LED. Obecnie stosuje się diody o standardowych kształtach (okrągłe, prostokątne) i wymiarach (3mm, 5mm). Bardzo duże zastosowanie mają diody dwukolorowe sterowane napięciem. Najczęściej montowane są pod kontrolkami sygnalizacyjnymi stan poszczególnych systemów.

Diody LED stanowią również podstawę wyświetlaczy siedmiosegmentowych wykorzystywanych do zobrazowania informacji w postaci cyfrowej, dostarczając tym samym informacji ilościowej. Wyświetlają wartości wprowadzonych przez operatora parametrów lub prezentują ich aktualne wartości. Szczególnie zauważalne są wskazania na modułach wskaźników, centralnego komputera.

Podobną rolę w panelach operatorskich i kabinach współczesnych pojazdów pełnią wyświetlacze ciekłokrystaliczne. W oparciu o matryce pasywne wyświetla się informacje tekstowe na pulpitach sterujących pracą właściwych podsystemów. Natomiast wyświetlacze ciekłokrystaliczne z matrycami aktywnymi stanowią podstawę tzw. szklanej kabiny. Zespół wskaźników obrazowych służy do komunikacji z centralnym komputerem pokładowym.

4 Ogólne zasady implementacji oprogramowania dla systemu mikroprocesorowego

W tym rozdziale

- Charakterystyka języków programowania mikrokontrolerów
- Narzędzia wspomagające programowanie i uruchamianie systemów mikroprocesorowych

4.1. Charakterystyka języków programowania mikrokontrolerów

Cechą charakterystyczną systemów mikroprocesorowych jest to, że składają się ze sprzętu i oprogramowania [10, 20]. Określenie „sprzęt" (ang. *hardware*) oznacza zamontowane w obudowie płytki z obwodami drukowanymi z umieszczonymi na nich układami scalonymi i elementami elektronicznymi. Określenie „oprogramowanie" (ang. *software*) oznacza zestaw instrukcji i danych, czyli program przeznaczony do wykonania przez mikroprocesor. Oprogramowanie jest nieodłączną częścią każdego komputera, decydującą o jego cechach użytkowych w stopniu nie mniejszym (a często większym) niż sprzęt. Możliwość tworzenia różnego rodzaju programów i wykonywania tych programów na tym samym sprzęcie zadecydowała o tak szybkim rozwoju komputerów cyfrowych i ich zastosowań m.in. w urządzeniach mechatronicznych. Program, który wykonuje procesor jest zapisem algorytmu w określonym języku programowania. Projektowanie i tworzenie programów komputerowych nazywamy programowaniem komputerów, albo krótko programowaniem.

Języki programowania komputerów wbudowanych

Językiem programowania nazywamy sformalizowany język służący do zapisu algorytmów. Języki sformalizowane są językami sztucznymi, tzn. utworzonymi drogą przyjęcia pewnych aksjomatów i definicji. Dla takiego języka podaje się alfabet i zbiór możliwych słów, ściśle określając zasady składni oraz jednoznacznie definiując reguły wyznaczające sens poszczególnych napisów. Języki programowania dzieli się przede wszystkim pod względem stopnia zaawansowania na:

- języki pierwszej generacji - są to języki maszynowe, czyli języki procesorów – jedyne języki „zrozumiałe" dla układów logicznych komputerów. Zarówno rozkazy jak i dane są w nich zapisywane w postaci liczb binarnych;
- języki drugiej generacji - języki symboliczne, asemblery. Są to języki niskiego poziomu, pod względem składni tożsame

z maszynowymi, z tą różnicą, że zamiast liczb używa się tu łatwiejszych do zapamiętania symbolicznych kodów rozkazów i adresów – mnemoników;

- języki trzeciej generacji - języki wysokiego poziomu, proceduralne. W językach tych jedna instrukcja jest tłumaczona na kilka rozkazów procesora, najczęściej od 5 do 10. Pierwszym językiem tego typu był ALGOL. Do tej grupy należą: FORTH, BASIC - języki niestrukturalne; Jovial, FORTRAN, Pascal, C, - języki strukturalne; Ada, C++, Java - języki zorientowane obiektowo;

- języki czwartej generacji - języki bardzo wysokiego poziomu, nieproceduralne. Korzystając z tych języków programista skupia się na problemie, a nie na sposobie jego rozwiązania, stąd nazwa języki zorientowane problemowo (ang. *task oriented languages*). Syntaktyka wielu języków czwartej generacji przypomina składnię języka naturalnego. Są one często używane do dostępu do baz danych. Przykładem języka z tej grupy jest SQL;

- języki piątej generacji - języki sztucznej inteligencji, języki systemów ekspertowych. Języki najbardziej zbliżone do języka naturalnego. Przykładem języka piątej generacji jest PROLOG.

Komputery pokładowe i systemy wbudowane maszyn i pojazdów są programowane z wykorzystaniem wymienionych wyżej języków programowania. W praktyce powszechnie stosuje się dwa rodzaje języków programowania systemów mikroprocesorowych.

Możliwość pierwsza polega na stosowaniu języka asemblera [2, 16, 27, 35, 38]. Jest to tzw. język niższego poziomu. Język ten operuje bezpośrednio na liście rozkazów danej jednostki centralnej. Są to operacje typu: „przenieść zawartość komórki o określonym adresie w pamięci zewnętrznej do określonego rejestru wewnętrznego jednostki centralnej". Programista nie pisze jednak programu bezpośrednio w kodzie maszynowym. Bezpośrednie posługiwanie się ciągami zer i jedynek byłoby zbyt uciążliwe. Dlatego w języku asemblera każdy rozkaz z listy rozkazów jednostki centralnej ma przyporządkowane krótką nazwę, tzw. mnemonik. Nazwa ta jest tak wybrana, aby kojarzyło się ono z funkcją danego rozkazu. Przykładowo rozkazy związane z przesunięciem komórek z jednego obszaru pamięci do drugiego mają najczęściej mnemoniki typu LD (ang. *load*) lub MOV (ang. *move*). Zadaniem programu tłuma-

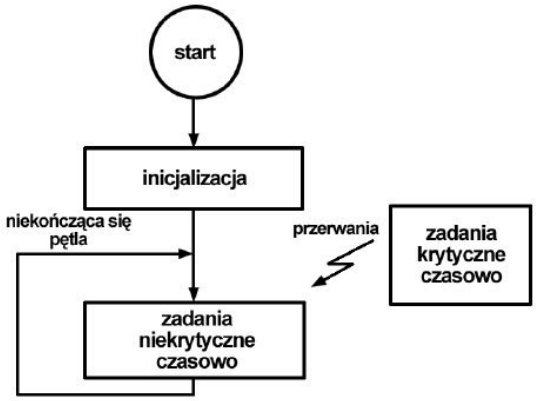
cącego jest proste przetłumaczenie mnemoników na odpowiadający im kod maszynowy. Program tłumaczący z języka asemblera określany jest po prostu jako asembler (ang. *assembler*)

Możliwość druga polega na stosowaniu języka programowania wyższego poziomu. Najczęściej stosowanymi obecnie językami z tej grupy jest język C [9, 23, 24, 28, 29, 44, 46] i jego rozwinięcie język C++ oraz język BASCOM [4, 5, 19, 45]. Języki programowania wyższego poziomu są w zasadzie niezależne od listy rozkazów konkretnej jednostki centralnej. Programista operuje rozkazami wykonującymi bardziej złożone operacje niż język asemblera. Przykładowo dla porównania dwóch liczb programista w języku C lub BASCOM używa bezpośrednio znaku relacji „>" lub „<". W języku asemblera ta prosta operacja musi być realizowana przy pomocy kilku lub nawet kilkunastu rozkazów w zależności od wielkości porównywanych liczb. Program tłumaczący z języka programowania wyższego rzędu na kod maszynowy nosi nazwę kompilatora (ang. *compiler*).

W programowaniu systemów mikroprocesorowych istnieje jednak wyraźna tendencja do wypierania programowania w języku asemblera przez programowanie w języku wysokiego poziomu. W szczególności dla mikroprocesorów i mikrokontrolerów o szerokości szyny danych od 16 bitów wzwyż język asemblera stosuje się już rzadko.

Przy pisaniu programu użytkownika należy zwrócić wagę na fakt, że poprawnie napisany program musi działać w niekończącej się pętli. Jak już wyjaśniono, jednostka centrala po włączeniu napięcia zasilania bez przerwy pobiera z pamięci i wykonuje kolejne rozkazy. Nie może się ona zatem po prosu zatrzymać po wykonaniu ostatniego rozkazu. Z tego powodu program użytkownika ma zawsze strukturę przedstawiona na rysunku 4.1.

Po włączeniu napięcia zasilania w programie użytkownika musi się odbyć jednorazowa inicjalizacja np. ustalenie trybu pracy urządzeń peryferyjnych. Właściwy program użytkownika jest realizowany najczęściej częściowo w niekończącej się pętli głównej oraz częściowo w przerwaniach.



Rysunek 4.1. Struktura programu użytkownika w systemie mikroprocesorowym

Język programowania BASCOM

Język Bascom [4, 5, 19, 45] wywodzi się z opracowanego w połowie lat 60 przez Johna Kemeny'ego i Thomasa Kurtza z Dartmouth College najprostszego języka programowania wysokiego poziomu BASIC (ang. *Beginner's All-Purpose Symbolic Instruction Code*). BASIC stał się popularny na początku lat 80 wraz z rozwojem komputerów osobistych i jest nadal stosowany w różnych odmianach. Za pomocą tego języka najłatwiej można poznać podstawowe pojęcia, charakterystyczne dla wszystkich języków programowania.

Zastosowany w środowisku programistycznym Bascom język przygotowania programów przypomina składnię języka BASIC. Jednak w języku Bascom znacznie zwiększono liczbę instrukcji związanych z mikrokontrolerami, co ułatwia pracę programiście i przyspiesza przygotowanie programu. Chociaż w języku Bascom nie ma ściśle określonej struktury pliku źródłowego przygotowywanego programu, to dla większej jego przejrzystości pisany program można podzielić na bloki zawierające deklaracje oraz definicje podprogramów, zmiennych, stałych, aliasów (nazw zamiennych) i blok głównej części programu.

Pojedyncza linia programu w języku Bascom może zawierać następujące części:

```
[[identyfikator:]][[instrukcja]][:instrukcja]] …
[['komentarz]]
```

W jednej linii może znajdować się kilka instrukcji oddzielonych znakiem dwukropka (:). Ale można też pisać każdą instrukcję w oddzielnym wierszu. Na przykład:

```
[[identyfikator:]]
[[instrukcja]] … [['komentarz]]
[[instrukcja]] … [['komentarz]]
```

Identyfikator, nazywany też etykietą, może składać się z dozwolonych liter lub cyfr i zawierać od 1 do 32 znaków. Musi zaczynać się literą, a kończyć dwukropkiem.

Linie programu w Bascomie mogą być wykonywalne albo niewykonywalne. Linie wykonywalne (**instrukcje**) tworzą działający program realizujący określony algorytm. Linie niewykonywalne tworzą cześć informacyjną oraz organizacyjną programu. Do najważniejszych linii tego rodzaju należą:

- linie komentarza - rozpoczynające się słowem `Rem` lub znakiem apostrofu (`'`);
- linie deklaracji - zawierające słowa `Dim`, `Declare`;
- linie dyrektyw kompilatora - rozpoczynające się znakiem dolara (`$`).

W środowisku BASCOM można zdefiniować stałe i zmienne różnych typów. Typ określa rozmiar zajmowanej pamięci (bit, bajt, słowo 16 bitowe itp.) oraz zakres wartości, które może przyjmować dana zmienna. Deklaracje zmiennych w programie wykonujemy za pomocą słowa kluczowego `Dim`:

```
Dim I As Intege
```

W języku Bascom są zdefiniowane następujące typy danych:

`BIT`	1 bit - przyjmuje stan 0 (fałsz) lub 1 (prawda);
`BYTE`	1 bajt -liczba całkowita bez znaku z zakresu 0..255;
`INTEGER`	2 bajty - liczba całkowita ze znakiem z zakresu: -32768 ..+32767;
`WORD`	2 bajty - liczba całkowita bez znaku z zakresu: 0...65535;

`LONG` 4 bajty - liczba zmiennoprzecinkowa ze znakiem z zakresu: -2147483648 .. +2147483647;

`SINGLE` 4 bajty - liczba stało lub zmiennoprzecinkowa pojedynczej precyzji;

`DOUBLE` 8 bajtów - liczba stało lub zmiennoprzecinkowa podwójnej precyzji;

`STRING` łańcuch znaków - (napis) o długości do 254 znaków.

We wszystkich obliczeniach prowadzonych w programie, a opisanych odpowiednimi wyrażeniami, oprócz nazw zmiennych używane są operatory. Wyrażenia zapisywane z użyciem dostępnych operatorów, można podzielić na:

- arytmetyczne, używane podczas obliczeń;
- relacji, używane podczas operacji porównywania, a także w zapisie warunków logicznych w pętlach i rozgałęzieniach;
- logiczne, używane przy formułowaniu warunków logicznych, pętli i manipulacji bitami.

W wyrażeniach mogą być stosowane stałe, zmienne i ich kombinacje połączone za pomocą operatorów arytmetycznych, relacji lub logicznych. Zbiór operacji arytmetycznych zawiera następujące operacje: +, -, *, \, / oraz ^. Należy rozróżniać dwa operatory dzielenia:

/ - jest operatorem dzielenia, w wyniku którego otrzymujemy liczbę zmiennoprzecinkową,

\ - jest operatorem dzielenia, w wyniku którego otrzymujemy liczbę całkowitą. Operator ten powinien być używany tylko dla zmiennych typu całkowitego (`Byte, Integer, Word, Long`).

Z dzieleniem całkowitym jest związany inny operator, tzw. operator reszty z dzielenia - `Mod`. Operator ten umożliwia obliczenie reszty z dzielenia całkowitego.

Przy porównywaniu wartości dwóch zmiennych lub stałych używane są operatory relacji, które stosowane są przede wszystkim w instrukcjach warunkowych mających wpływ na kolejność wykonywania instrukcji. W języku Bascom używane są następujące operatory relacji:

Operator	Funkcja	Wyrażenie
=	równość	X = Y
<>	nierówność	X <> Y
<	mniejszy	X < Y
>	większy	X > Y
<=	mniejszy lub równy	X <= Y
>=	większy lub równy	X >= Y

Operacje relacji zwracają wartość logiczną true (prawda, wartość 1), jeśli wyrażenie jest prawdziwe lub false (fałsz, wartość 0), jeśli wyrażenie jest fałszywe. W języku Bascom zdefiniowano również cztery operacje logiczne:

Operator	Znaczenie
Not	negacja
And	iloczyn logiczny
Or	suma logiczna
Xor	suma symetryczna

Do selektywnego wykonywania instrukcji, zależnego od spełnienia jakiegoś warunku logicznego lub przyjmowania określonej wartości przez zmienną selekcyjną wykorzystywane są instrukcje warunkowe i wyboru. Dzięki tym instrukcjom można wybierać, które z sekwencji instrukcji będą wykonane. W Bascom jest dostępna instrukcja warunkowa `If…Then` i instrukcja wyboru `Select…Case`.

Składnia instrukcji If…Then jest następująca:

```
If wyrazeniel Then ciag_instrukcji_a
[Else ciag_instrukcji_c]
```

Oraz w postaci złożonej:

```
If wyrazeniel Then
    ciąg_instrukcji_a
[Else if wyrazenie2 Then ciag_instrukcji_b]
[Else ciag_instrukcji_c]
End If
```

gdzie:

`wyrazeniel, wyrazenie2` - wyrażenia określające warunki logiczne,

`ciag_instrukcji_a` - ciąg instrukcji wykonywanych, gdy spełniony jest warunek określony przez `wyrazeniel`,

ciag_instrukcji_b - ciąg instrukcji wykonywanych, gdy spełniony jest warunek określony przez wyrażenie2,

ciag_instrukcji_c - ciąg instrukcji wykonywanych, gdy niespełniony jest warunek określony przez wyrażenie1 oraz wyrażenie2.

W większości programów trudno obejść się bez instrukcji lub bloku (instrukcji), które muszą być wielokrotnie powtórzone. Do tworzenia takich powtórzeń służą pętle. W Bascomie są dostępne trzy rodzaje instrukcji pętli: For…Next, Do…Loop oraz While…Wend.

W instrukcji pętli For…Next blok instrukcji zawarty pomiędzy słowem For a Next, jest wykonywany określoną liczbę razy z jednoczesną modyfikacją wartości zmiennej określającej liczbę powtórzeń. Składnia instrukcji For…Next jest następująca:

```
For zmienna = liczba_pocz To liczba_kon [Step krok]
Instrukcje…
Next [zmienna]
```

gdzie:

zmienna - zmienna numeryczna, będąca licznikiem przejść pętli (wskaźnikiem pętli),

liczba_pocz - liczba początkowa licznika pętli,

liczba_kon - liczba końcowa licznika pętli,

krok - krok zmiany wartości licznika pętli po osiągnięciu instrukcji Next

instrukcje - ciąg instrukcji.

Często wykorzystywaną instrukcją pętli jest instrukcja Do…Loop. Blok instrukcji jest w niej powtarzany dopóty, dopóki warunek końcowy nie będzie spełniony. Dla tej instrukcji warunek końcowy jest opcjonalny i jeżeli go nie ma, pętla będzie pętlą nieskończoną. Składnia instrukcji jest następująca:

```
Do
Instrukcje…
Loop [Until warunek]
```

gdzie:

`warunek` - warunek opuszczenia pętli,

`instrukcje` - ciąg instrukcji.

Instrukcje zawarte pomiędzy `Do` a `Loop`, będą wykonywane, dopóki nie będzie spełniony warunek po słowie `Until`. Spełnienie warunku zakończy wykonywanie pętli. Ponieważ warunek w tej pętli jest sprawdzany na końcu, instrukcje zawarte w pętli zawsze zostaną wykonane co najmniej jeden raz. Gdy warunek nie jest podany, to jak wspomniano, pętla będzie wykonywana w nieskończoność. Można wyjść z pętli w dowolnym momencie stosując instrukcję `Exit Do`, tak jak w instrukcji pętli `For…Next`.

W pętli `While` jest wykonywany określony ciąg instrukcji dopóty, dopóki warunek jest spełniony. Jest to pętla podobna działaniem do pętli `Do…Loop`, z tą różnicą, że warunek jest obliczany przed wejściem do pętli - na jej początku. Składnia tej instrukcji pętli jest następująca:

```
While  warunek
Instrukcje…
Wend
```

gdzie:

`warunek` - wyrażenie określające warunek logiczny pętli,

`instrukcje` -ciąg instrukcji.

Gdy podany w pętli warunek jest spełniony (prawdziwy), wtedy jest wykonywany ciąg instrukcji umieszczony pomiędzy `While` a `Wend`. Wyjście z pętli jest możliwe tylko wtedy, gdy warunek nie jest spełniony lub za pomocą instrukcji `Exit While`, tak jak dla opisanych wyżej instrukcji pętli. Instrukcje w tej pętli mogą nie być w ogóle wykonane, nie tak jak w pętli `Do…Loop`, w której wykonywane są co najmniej jeden raz.

W języku Bascom można z wyodrębnionych części programu budować podprogramy, procedury i funkcje.

Podprogramy są to fragmenty programu, które mogą być wykorzystywane wielokrotnie. Podprogramów nie trzeba wcześniej deklarować, gdyż nie mają żadnych parametrów, dla których kompilator musiałby zarezerwować pamięć. Składnia podprogramu jest następująca:

```
Etykieta:
Instrukcje podprogramu
Return
```

gdzie:

`Etykieta` - nazwa etykiety podprogramu, do której następuje skok z dowolnego miejsca programu

W podprogramie `etykieta` jest nazwą (identyfikatorem) podprogramu, do którego następuje skok, a po wykonaniu podprogramu powrót do następnej instrukcji programu głównego, po instrukcji wywołania podprogramu. Podprogramy powinny być umieszczane w końcowej części programu, po instrukcjach programu głównego. Wywoływane podprogramy muszą kończyć się instrukcją `Return`. Wtedy nastąpi powrót do instrukcji następnej po instrukcji wywołania podprogramu (dla podprogramów instrukcją wywołującą podprogram jest instrukcja `Gosub`). Instrukcja Return nie służy tylko do zakończenia podprogramów, ale także do zakończenia podprogramów obsługi przerwań.

Procedurą nazywa się wyodrębnioną część programu, która ma swoją nazwę, realizuje określone zadania i w ustalony sposób komunikuje się z programem głównym. Procedury stosuje się do wykonywania czynności wielokrotnie powtarzanych w programie. Stosowanie procedur przyczynia się do bardziej efektywnego wykorzystania pamięci i większej przejrzystości programu. Zastosowane w programie procedury muszą być wcześniej zadeklarowane. Deklaracje procedur zazwyczaj umieszcza się na początku programu. Składnia instrukcji deklarującej procedurę użytkownika jest następująca:

```
Declare Sub nazwa [([Byref|Byval] parametr As
typ,[Byref|Byval] parametr As typ...)]
```

gdzie:

`nazwa` - nazwa deklarowanej procedury,

`parametr` - nazwa parametrów formalnych,

`typ` - typ parametrów formalnych deklarowanej procedury, przekazywanych do niej podczas wywołania.

Każda procedura musi być zadeklarowana instrukcją `Declare Sub` przed jej pierwszym użyciem w programie. Jest to konieczne, aby kompilator określił, jakie parametry powinny być przekazane procedurze i

zarezerwował dla nich odpowiedni obszar pamięci. Parametry występujące w deklaracji procedury są opcjonalne.

Funkcja jest specyficzną postacią procedury, z możliwością przekazywania poprzez jej nazwę wartości określonego typu, zgodnego z typem zadeklarowanym w nagłówku funkcji. Funkcje, podobnie jak procedury, muszą być deklarowane w początkowej części programu. Składnia deklaracji funkcji jest następująca:

```
Declare Function nazwa [([Byref|Byval] parametr As
typ,[Byref|Byval] parametr As typ...)] As typ_zwr
```

gdzie:

`nazwa` - nazwa deklarowanej funkcji,

`parametr` - nazwa parametru (lub nazwy parametrów),

`typ` - typ parametrów formalnych deklarowanej funkcji,

`typ_zwr` - typ wartości zwracanej przez funkcję.

Każda funkcja musi być zadeklarowana po słowach kluczowych `Declare Function` przed jej pierwszym użyciem w programie. Jest to konieczne, aby kompilator zarezerwował miejsce w pamięci mikrokontrolera na parametry przekazane funkcji oraz wynik zwracany przez funkcję.

Jednym z elementów języka Bascom są dyrektywy. Poprzez zastosowanie dyrektyw preprocesora jest możliwe wykonanie kompilacji warunkowej (na przykład tylko wybranych fragmentów programu) oraz przekazanie informacji kompilatorowi o programowanym mikrokontrolerze, częstotliwości oscylatora taktującego mikrokontroler, szybkości transmisji przez UART/USART i o wielu innych parametrach związanych z programem i docelowym mikrokontrolerem. Ponieważ każdy mikrokontroler ma układy peryferyjne, które przed użyciem należy skonfigurować, język Bascom wyposażono w szereg instrukcji konfigurujących. Te instrukcje są przeznaczone nie tylko do konfigurowania układów wewnętrznych mikrokontrolera, ale także układów zewnętrznych, które do niego dołączono (wyświetlacz LCD, odbiornik podczerwieni itp.).

Bascom pozwala również na wstawianie do programu wstawek asemblerowych, które umożliwią pełną kontrolę nad generowanym kodem wynikowym oraz czasem jego wykonywania.

4.2. Narzędzia wspomagające programowanie i uruchamianie systemów mikroprocesorowych

Projektowanie, uruchamianie i testowanie części sprzętowej i oprogramowania systemów mikroprocesorowych jest procesem pracochłonnym, złożonym i kosztownym – wymaga stosowania specjalizowanych narzędzi tzw. środków uruchomieniowych (ang. *Development Systems*). Środki te wydatnie usprawniają zarówno proces projektowania, jak i zarządzania projektem, śledzenia postępów czy wreszcie testowanie kompletnego systemu mikroprocesorowego. Można je podzielić na:

- sprzętowe,
- sprzętowo-programowe,
- programowe.

Środki sprzętowe

Uruchomienie części sprzętowej systemu mikroprocesorowego wiąże się z koniecznością sprawdzenia:

- poprawności działania poszczególnych układów elektronicznych;
- poprawności połączeń elektrycznych;
- krok po kroku poprawnego przechodzenia sygnałów przez poszczególne układy;
- działania modułów w czasie rzeczywistym oraz sprawdzenia poprawności wzajemnego komunikowania się modułów.

Czynności te mogą być wykonane za pomocą ogólnodostępnych, uniwersalnych środków sprzętowych w postaci:

- multimetrów;
- oscyloskopów;

- sond logicznych;
- dołączanych pulpitów operatora.

Środki sprzętowo-programowe

Do tej klasy środków zaliczamy nowoczesne narzędzia wspomagające uruchomianie systemów mikroprocesorowych, które wykorzystują komputery osobiste. Takie podejście eliminuje potrzebę stosowania złożonych i drogich autonomicznych urządzeń analizujących, wymaga jedynie zespołów dopasowujących, synchronizujących pracę i zapewniających wymianę danych. Całość komunikacji z otoczeniem i analizę pracy systemu przejmuje oprogramowanie komputera osobistego. Tego typu systemy uruchamiające są wyposażone w moduł emulacji sprzętowej. Połączenie tego modułu z uruchomionym systemem realizuje się za pomocą odpowiedniej wtyczki, wchodzącej w podstawkę mikroprocesora lub pamięci uruchamianego systemu. Dzięki temu uzyskujemy możność obserwacji pracy uruchamianego systemu z wykorzystaniem wszystkich środków sprzętowych i programowych dostępnych w systemie mikrokomputerowym.

Do tej klasy urządzeń zalicza się analizatory stanów logicznych i uniwersalne programatory pamięci, mikrokontrolerów i programowanych struktur logicznych podłączanych do komputera nadrzędnego, za pośrednictwem standartowego interfejsu szeregowego RS232C.

Środki programowe

Do tej grupy środków uruchomieniowych zalicza się oprogramowanie narzędziowe pozwalające na projektowanie i testowania oprogramowania użytkowego dla systemu mikroprocesorowego oraz inne oprogramowanie dodatkowe nie związane bezpośrednio z tworzeniem oprogramowania a wykorzystywane do testowania np. poprawności transmisji.

Oprogramowanie systemu mikroprocesorowego jednoznacznie określa sposób działania danego systemu. W jego skład wchodzą dwa typy programów:

- **oprogramowanie podstawowe**, stanowiące zbiór programów uzupełniających sprzęt systemu o zasadnicze funkcje umożliwiające wykorzystanie tego sprzętu przez inne programy;

- **oprogramowanie użytkowe**, stanowiące zespół programów realizujących funkcje użytkowe systemu określone na etapie projektu koncepcyjnego, wykorzystując własności sprzętowe systemu mikroprocesorowego za pomocą i pośrednictwem funkcji dostarczanych przez oprogramowanie podstawowe.

Zadaniem oprogramowania podstawowego jest więc stworzenie pomostu między sprzętem a oprogramowaniem użytkowym poprzez udostępnienie mu pewnych funkcji związanych najczęściej z efektywnym wykorzystaniem możliwości sprzętowych systemu. Do oprogramowania podstawowego zalicza się różnego rodzaju **systemy operacyjne**. Jednym z nich jest **system operacyjny czasu rzeczywistego** [41, 43]. Jest to system operacyjny gwarantujący reakcję na określone zdarzenia zewnętrzne w nieprzekraczalnym czasie. W przypadku systemów mikroprocesorowych pod pojęciem systemu operacyjnego rozumie się również **oprogramowanie rezydentne**, umożliwiające ładowanie do pamięci i uruchamianie programów w systemie docelowym. Z punktu widzenia złożoności oprogramowania rezydentnego dzieli się je czasami na dwie grupy, wydzielając poza systemami operacyjnymi także **monitory**. Przez monitor rozumie się najprostszą wersję systemu operacyjnego składającą się ze zbioru procedur obsługi podstawowych układów wejścia-wyjścia (interfejs szeregowy, klawiatura).

Zadaniem oprogramowania użytkowego jest zrealizowanie zadań nakładanych na dany komputer na etapie projektu koncepcyjnego. Tworzenie oprogramowania użytkowego dla systemów mikroprocesorowych (systemów wbudowanych) opartych na mikroprocesorach i dedykowanych mikrokontrolerach znacznie się różni od tworzenia oprogramowania dla tradycyjnych systemów obliczeniowych. Ten typ tworzenia oprogramowania określa się w literaturze mianem **programowania zagnieżdżonego** (ang. *embdeded programming*) [10]. Główne cechy programów zagnieżdżonych są następujące:

- program jednoznacznie ustala funkcję systemu mikroprocesorowego, tzn. użytkownik ma możliwość zmiany funkcji systemu zazwyczaj tylko w niewielkim zakresie przewidzianym przez program użytkowy (np. jeśli sterownik realizuje funkcję regulatora temperatury i obrotów silnika, użytkownik może wprowadzić nowe nastawy regulatora, lecz nie może zmienić całkowicie funkcji urządzenia). Jest to cecha określana jako „zagnieżdżenie" programu;

- działanie programu musi spełniać określone wymagania czasowe dotyczące nie przekraczania maksymalnego czasu

reakcji na określone zdarzenia zewnętrzne oraz realizację określonych zadań programowych w skończonym i nieprzekraczalnym czasie (np. w przytoczonym przykładzie regulatora, pomiar temperatury i obrotów, obliczenie sygnału sterującego oraz jego wydanie musi odbywać się w ściśle określonych odstępach czasu, co warunkuje założoną jakość procesu regulacji). Ta cecha określana jest jako praca „w czasie rzeczywistym";

- są to programy działające na specyficznych zasobach sprzętowych warunkowanych budową sprzętową systemu mikroprocesorowego na konkretne zadanie (np. w podanym przykładzie regulatora ważnym elementem programu będzie obsługa przetwornika analogowo-cyfrowego i układu licznika impulsów).

Do produkcji tego typu oprogramowania wykorzystuje się specjalne programy zaliczane do tzw. **oprogramowania wspomagającego** (lub **narzędziowego**), obejmujące następujące grupy programów:

- edytory tekstów, programy redagujące;
- kompilatory języków programowania (ang. *compilers*);
- programy wspomagające uruchamianie (ang. *debuggers*);
- programy łączące (ang. *linkers*);
- programy zarządzające bibliotekami (ang. *library managers*).

Programy narzędziowe służące do tworzenia oprogramowania dla procesorów specjalizowanych - mikrokontrolerów łączone są zwykle w jeden pakiet programów i dostarczane są przez wyspecjalizowanego producenta, podobnie jak i elementy oprogramowania podstawowego (np. systemy operacyjne). W tej grupie programów znajduje się również oprogramowanie do uruchamiania programu w systemie mikroprocesorowym. Pozwala ono na eliminację błędów i wprowadzanie zmian do programu użytkowego w celu uzyskania bezbłędnej i zgodnej z założeniami pracy systemu. Tak powstałe pakiety oprogramowania służące do tworzenia, modyfikowania, testowania i konserwacji oprogramowania noszą nazwę **zintegrowanych środowisk programistycznych i uruchomieniowych** IDE (ang. *Integrated Development Environment*) [2, 9, 10, 11, 23, 28, 29, 45, 46]. W tych środowiskach powszechnie stosuje się dwie metody uruchamiania programów dla mikrokontrolerów - bez wy-

korzystania i z wykorzystaniem systemu docelowego. W pierwszym przypadku wykorzystuje się symulatory programowe a w drugim emulatory sprzętowe pamięci i procesora oraz monitory programowe.

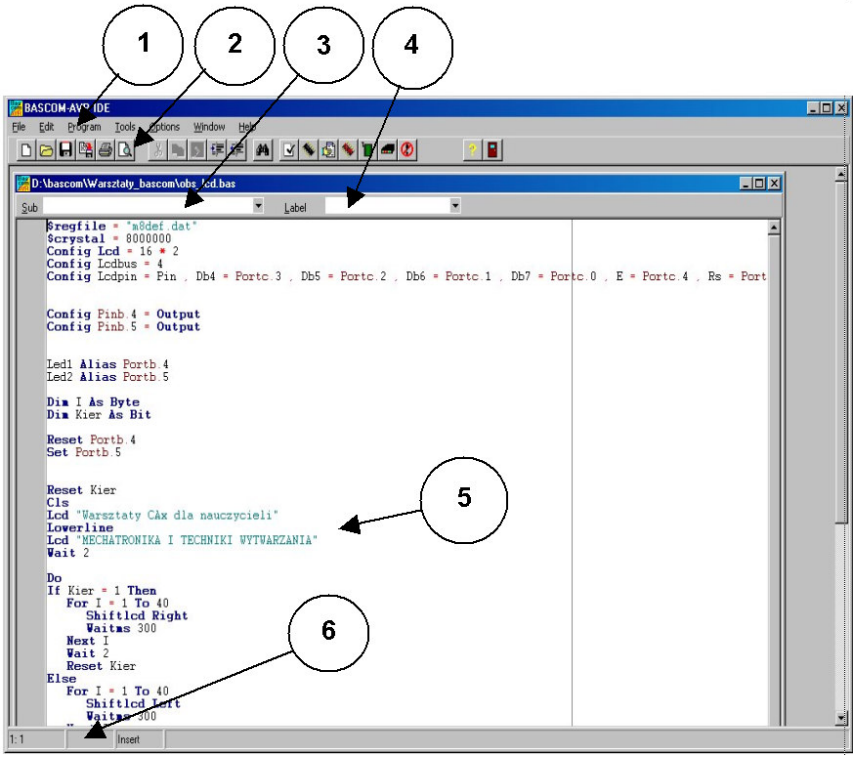
Środowisko programistyczne i uruchomieniowe BASCOM AVR

W skład pakietu Bascom AVR [3], tworzącego środowisko programistyczne i uruchomieniowe dla mikrokontrolerów rodziny AVR, wchodzi kilka aplikacji służących między innymi do symulacji programów i programowania mikrokontrolerów. W pakiecie znajdują się również programy: terminala i edytora znaków dla wyświetlacza LCD oraz wiele innych programów ułatwiających pracę programiście. Tworzone za pomocą wbudowanego w pakiet edytora programy są zapisywane w plikach z rozszerzeniem .bas. Podczas pisania programu w Bascom AVR wskazane jest korzystanie z wbudowanej doskonałej pomocy (help) [4, 5], w której znajduje się opis wszystkich instrukcji zilustrowany przykładami ich użycia.

Pakiet oprogramowania Bascom jest dosyć rozbudowanym zintegrowanym pakietem oprogramowania, ale nie oznacza to, że jest trudny do przyswojenia i stosowania. Wprost przeciwnie, jest programem niezwykle łatwym w użyciu i bardzo funkcjonalnym, a jego obsługa nie sprawia trudności, ponieważ jest typowym programem pracującym pod Windows. Większość poleceń w pasku menu jest taka sama, jak w innych programach Windows. Za pomocą oprogramowania Bascom AVR możliwe jest szybkie i łatwe przygotowywanie programów dla większości mikrokontrolerów AVR bez dokładnej znajomości ich wbudowanych układów peryferyjnych oraz rejestrów. Bascom wyposażono w wiele gotowych instrukcji (procedur i funkcji) do obsługi wyświetlaczy LCD (alfanumerycznych i graficznych), interfejsów I2C, 1-Wire czy RS232 i wielu innych układów peryferyjnych.

Obsługę programu ułatwiają skróty klawiszowe oraz menu kontekstowe, wywoływane prawym klawiszem myszki, zawierające najczęściej używane w danym kontekście polecenia oraz nazwy wywoływanych programów.

Wersję demonstracyjną pakietu można pobrać bezpłatnie ze strony producenta www.mcselec.com. Instalacja pakietu jest przeprowadzana w sposób typowy dla wielu innych programów. Za pomocą wersji demonstracyjnej można kompilować program o kodzie wynikowym do 2 kB. Nie jest to dużo, ale pozwala na napisanie wielu użytecznych programów.



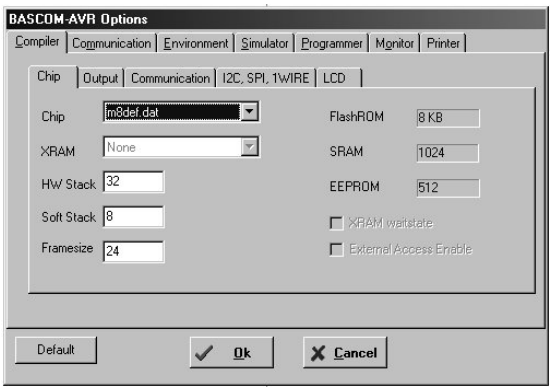
Rysunek 4.2. Widok okna głównego pakietu Bascom AVR

Po zainstalowaniu oprogramowania w oknie głównym środowiska Bascom AVR (rysunek 4.2) można wyróżnić następujące składowe:

1. Pasek menu.
2. Pasek narzędziowy składający się z ikon najczęściej używanych programów oraz poleceń.
3. Rozwijana lista zdefiniowanych procedur (Sub) umożliwiająca szybki skok do wybranej procedury w pisanym programie.
4. Rozwijana lista etykiet (Label) w pisanym programie umożliwiająca szybki skok do wybranej etykiety.
5. Obszar roboczy edytora przeznaczony do pisania programu.
6. Pasek statusu.

Przed napisaniem pierwszego programu w języku Bascom należy wykonać podstawową konfigurację środowiska uruchomieniowego. Wybór

ustawień konfiguracyjnych przeprowadza się w oknie dialogowym otwieranym z menu Options, w którym w pierwszej kolejności można skonfigurować kompilator. Parametry konfigurujące poszczególne programy pakietu są ustalane na odpowiadających im zakładkach przedstawionych na rysunku 4.3.



Rysunek 4.3. Okno dialogowe konfigurowania kompilatora po wybraniu zakładki Chip

Zakładka Chip w oknie dialogowym Compiler umożliwia ustalenie parametrów związanych z przewidywanym do zastosowania mikrokontrolerem. Każdy mikrokontroler ma zapisaną swoją specyfikację rejestrów w pliku *.dat.

W zakładce Output okna Compiler wybieramy pliki, jakie powinny być tworzone podczas kompilacji. Pliki te są niezbędne do prawidłowej pracy symulatora i programatora.

W zakładce Communication okna Compiler można ustawić wartości dwóch parametrów:

- Baudrate (np. 9600) - szybkości transmisji danych poprzez interfejs szeregowy RS232,
- Frequency (np. 8000000) - częstotliwości podstawowej oscylatora taktującego mikrokontroler.

Zakładka I2C, SPI, 1WIRE w oknie Compiler umożliwia przypisanie liniom komunikacyjnym interfejsów SPI, I2C oraz 1-Wire wyprowadzeń portów występujących w wybranym mikrokontrolerze.

Ostatnia zakładka menu konfiguracyjnego kompilatora - zakładka LCD pozwala określić parametry dotyczące obsługi wyświetlaczy LCD z popularnym kontrolerem HD44780.

Pozostałe zakładki okna dialogowego otwieranego z menu Options mogą pozostać na ustawieniach domyślnych, gdyż są w większości przypadków wystarczające do prawidłowej pracy pakietu i nie muszą być zmieniane z wyjątkiem zakładki Programmer. Zakładka ta umożliwia wybranie z rozwijanej listy jednego z obsługiwanych programatorów koniecznych do zaprogramowania mikrokontrolera. Po wybraniu określonego programatora jest samoczynnie wybierana zakładka przeznaczona do skonfigurowania jego portu komunikacyjnego.

Po wykonaniu wymienionych wyżej czynności konfiguracyjnych można przystąpić do napisania i kompilacji opracowywanego programu. Aby rozpocząć pisanie programu należy wybrać z menu File polecenie New lub nacisnąć klawisze Ctrl+N. Po napisaniu programu należy go skompilować wybierając z menu Program polecenie Compile lub nacisnąć klawisz F7. Czynność tą nożna również wykonać wybierając właściwą ikonkę graficzną z paska narzędziowego. W podobny sposób można obsługiwać pozostałe programy składowe pakietu, jak na przykład symulator, program sterujący programatorem i inne.

5 Przykłady programów obsługi wybranych układów peryferyjnych

W tym rozdziale

- Opis zestawu uruchomieniowego
- Przykłady programów obsługi wewnętrznych układów peryferyjnych mikrokontrolerów i standardowych urządzeń wejścia-wyjścia

5.1. Opis zestawu uruchomieniowego

Do nauki programowania systemów mikroprocesorowych niezbędny jest odpowiedni sprzęt w postaci układu uruchomieniowego z odpowiednim mikroprocesorem i takimi układami wejścia-wyjścia i urządzeniami peryferyjnymi, które umożliwią także komunikowanie się systemu ze światem zewnętrznym. Zaletą układu uruchomieniowego powinna być możliwość budowania prostych układów do sprawdzenia swojego pomysłu bez użycia lutownicy.

Wymienione wymagania spełniają układy uruchomieniowe o oznaczeniu ZL2AVR produkowane przez firmę KAMAMI. Zestawy umożliwiają opracowywanie własnych projektów, które będzie można przetestować, zanim zostanie dla nich zaprojektowana płytka drukowana, ponieważ potrzebne połączenia mogą być wykonywane specjalnymi przewodami zakończonymi miniaturowymi wtykami. Ponadto polecane układy uruchomieniowe z powodzeniem mogą być wykorzystane przy przygotowywaniu programów, a także uczenia się programowania mikrokontrolerów AVR w innych niż Bascom środowiskach programistycznych, takich jak C czy asembler.

W skład zestawu uruchomieniowego ZL2AVR wchodzą następujące elementy:

- **układ uruchomieniowy ZL2AVR** [49] - system mikroprocesorowy z mikrokontrolerem z rodziny AVR ATmega8 wykonany w oparciu o dwustronną płytkę drukowaną;
- **programator ISP ZL2PRG** [50] - uniwersalny programatorem ISP dla mikrokontrolerów AVR, wyposażony w interfejs umożliwiający programowanie pamięci już po zamontowaniu w systemie (ISP), zbudowany w oparciu o dwustronną płytkę drukowaną, a jego dodatkowym atutem jest możliwość współpracy z wieloma bezpłatnymi programami sterującymi jego pracą, m.in. programator obsługiwany jest przez środowisko programistyczne Bascom AVR;

- **alfanumeryczny wyświetlacz LCD** – jego zastosowanie umożliwia prezentację danych zobrazowujących wyniki działania opracowywanego oprogramowania;
- **zasilacz AC/DC o napięciu wyjściowym 9... 12 V** do zasilania całego układu.

Dokumentację do zestawu ZL2AVR można pobrać ze strony internetowej: http://www.btc.pl/index.php?id=zl2avr. Widok układu uruchomieniowego przedstawiono na rysunku 5.1.

Rysunek 5.1. Układ uruchomieniowy ZL2AVR

Na płytce układu uruchomieniowego umieszczone są elementy, z których można utworzyć wiele różnych układów mikroprocesorowych, bez konieczności lutowania. Na płytce znajdują się następujące elementy:

1. **Mikrokontroler ATmega8**, który może być taktowany wewnętrznym sygnałem zegarowym lub z zastosowaniem zewnętrznego rezonatora kwarcowego X1, dołączonego przez zworki JP2, JP3. Wszystkie wyprowadzenia mikrokontrolera są dostępne na podwójnych końcówkach a elementy L1, C3, C4, tworzą filtr napięcia zasilającego wewnętrzny przetwornik A/C.
2. **Układ zerowania mikrokontrolera** zbudowany z przycisku S5, zwory JP1 umożliwiającej dołączenie zewnętrznego obwodu zerującego do mikrokontrolera oraz rezystora R24 i kondensatora C5. W mikrokontrolerze linia zerowania RST może być wykorzystana jako zwyczajna linia I/0 portu C.

3. **Złącze Z7** do podłączania programatora ISP.
4. **Cztery wyświetlacze siedmiosegmentowe LED ze wspólną anodą** przeznaczone są do prezentowania danych liczbowych.
5. **Złącze interfejsu szeregowego RS232** zrealizowane z użyciem konwertera poziomów **MAX232** umożliwiające komunikację z komputerem PC oraz innymi układami mikroprocesorowymi.
6. **Układ PCF8574** pełniący funkcję konwertera I2C na 8-bitowy port I/0, który może pracować zarówno z liniami wejściowymi, jak i wyjściowymi, również w trybie mieszanym. Adresy konwertera są ustalone na stałe. Układ pozwala na wykonywanie eksperymentów z wykorzystaniem interfejsu I2C oraz zwiększa liczbę portów o dodatkowy port 8-bitowy.
7. **Złącza Z5 i Z6**, które umożliwiają dołączenie do układu zewnętrznych urządzeń sterowanych magistralą I2C z **rezystorami R27 i R28** koniecznymi do poprawnej pracy magistrali.
8. **Układ ULN2803A** pełniący rolę portu wyjściowego dużej mocy umożliwiającego sterowanie urządzeniami pobierającymi znaczny prąd.
9. **Diody sygnalizacyjne LED (D1...D8)** do sygnalizacji zdarzeń stwierdzonych przez zaprojektowany układ mikroprocesorowy i **rezystory R1…R8**, które ograniczają prądy płynące przez diody LED.
10. **Złącze Z10** umożliwiające dołączanie elementów sterowanych za pomocą interfejsu 1-Wire.
11. **Odbiornik transmisji danych w podczerwieni** umożliwiający sterowanie układem mikroprocesorowym za pomocą pilota.
12. **Złącze Z11 (Servo)**, które służy do dołączania mechanizmu modelarskiego.
13. **Przyciski ogólnego przeznaczenia S1...S4**, które w zależności od wartości napięcia podawanego na wejścia portów mikrokontrolera będą po ich przyciśnięciu podawać poziom niski, bądź wysoki.
14. **Złącze Z1** umożliwiające dołączenie do płytki klawiatury PC ze złączem PS2.

15. **Złącza Z4, Z8, Z9 (AUX)** pozwalające na wyprowadzenie napięcia zasilającego płytkę oraz **linie zasilające A1-A6**, które umożliwiają dołączenie do płytki elementów zewnętrznych.
16. **Potencjometr P2**, który umożliwia podanie na wejście przetwornika A/C zawartego w mikrokontrolerze napięcia z zakresu 0 do 5 V.
17. **Podstawka U2** do zamontowania dowolnego układu 14-pinowego.
18. **Podstawka U3** do zamontowania dowolnego układu 16-pinowego.
19. **Złącze W5** do podłączenia **wyświetlacza LCD** o organizacji 2x16 znaków umożliwiającego prezentacje danych w postaci alfanumerycznej.

Układy zawarte na płytce zasilane są napięciem +5 V stabilizowanym za pomocą stabilizatora LM7805. Dioda LED sygnalizuje włączenie zasilania płytki uruchomieniowej. Podczas wykonywania przykładów opisanych w dalszej części, konieczne będzie wykonanie za pomocą przewodów właściwych połączeń na płytce uruchomieniowej zgodnie ze schematem zamieszczonym w opisie.

5.2. Przykłady programów obsługi wewnętrznych układów peryferyjnych mikrokontrolerów i standardowych urządzeń wejścia-wyjścia

W tej części zostaną przedstawione przykłady programów napisanych w języku Bascom [5, 45], ilustrujące obsługę nie tylko wbudowanych układów peryferyjnych mikrokontrolera, ale także zewnętrznych układów i urządzeń wejścia-wyjścia. Wszystkie programy zostały spraw-

dzone z wykorzystaniem zestawu uruchomieniowego z mikrokontrolerem ATmega8 [1, 2]. Przykłady programów dobrano tak, aby zilustrować sposoby obsługi układów peryferyjnych mikrokontrolera: portów, układów czasowych, przetworników A/C a także elementów zewnętrznych dołączanych do mikrokontrolera takich jak: wyświetlacze i przyciski (klawiatury). Do przetestowania tych przykładów można wykorzystać układ uruchomieniowy (ZL2AVR), ale i bez tego zestawu można także sprawdzić działanie przykładowych programów za pomocą symulatora programowego, w jaki wyposażony jest Bascom.

Przykład obsługi portów wyjściowych

Pierwszy przykład zilustruje pracę portów oraz ich linii skonfigurowanych jako wyjścia. Każdy port oraz pojedyncza linia portu mikrokontrolera AVR może być skonfigurowana zarówno jako wejście jak i wyjście [1, 2, 11, 19, 23, 45, 46]. Do konfiguracji portów jako linii wejścia lub wyjścia służy specjalny rejestr DDRx. Bascom umożliwia ustalenie trybu pracy portów oraz ich linii za pośrednictwem instrukcji konfiguracyjnych Config Port dla całych portów oraz Config Pin dla pojedynczych linii portów. Składnia tych instrukcji jest następująca:

```
Config Portx = tryb    'ustawia kierunek transmisji
                       'portu x
Config Pinx.y = tryb   'ustawia kierunek transmisji
                       'wybranej linii portu
```

gdzie:

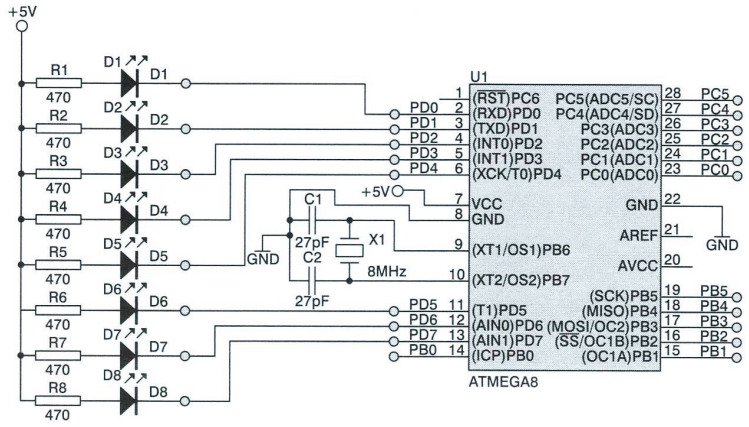
`tryb` – możliwy jest: Input, gdy linie portu mają być

wejściami lub `Output`, gdy linie portu mają być wyjściami;

`x` – nazwa portu;

`y` – numer linii portu (0…7).

W prezentowanym przykładzie sterowanych jest 8 diod dołączonych katodami do linii portu D mikrokontrolera - schemat dołączenia diod przedstawiono na rysunku 5.2.



Rysunek 2.2. Schemat dołączenia diod do mikrokontrolera

Ponieważ diody dołączono do portu D katodami, więc będą zapalane przy niskich poziomach napięcia na wyprowadzeniach tego portu. Dołączone szeregowo rezystory ograniczają prądy płynące przez każdą diodę do bezpiecznej wartości. Efekt węża świetlnego uzyska się przez kolejne (z opóźnieniem) załączanie co drugiej diody LED. Na wydruku 5.1 przedstawiono program sterujący diodami dołączonymi do portu D mikrokontrolera AVR.

Wydruk 5.1. Program sterowania diodami węża świetlnego:

```
'Program węża świetlnego sterującego 8-mioma diodami
'LED dołączonymi do portu D mikrokontrolera
$REGFILE = "m8def.dat" 'informuje kompilator o pliku
   'dyrektyw mikrokontrolera
$CRYSTAL = 8000000    'informuje kompilator
                      'o częstotliwości oscylatora
                      'taktującego mikrokontroler
Config Portd = Output 'wszystkie linie portu D jako
                      'wyjściowe
Portd = &B01010101    'wartość początkowa wpisana do
                      'portu wyjściowego D
Do                    'początek pętli nieskończonej
Rotate Portd, Left    'przesuwaj wpisane- wartości do
                      'portu D w lewo
Waitms 200      'opóźnienie przesunięć o 200 ms
Loop            'koniec pętli głównej programu
End             'koniec programu
```

Program powoduje zapalenie kolejno po 4 diody LED. Na przemian zapalane są diody o numerach parzystych i nieparzystych. Najpierw

jednak cały port D mikrokontrolera jest konfigurowany jako wyjściowy, oczywiście po użytych wcześniej dwóch dyrektywach kompilatora, które zawsze powinny się znaleźć w programie. Następnie do portu D zostaje wpisana wartość początkowa (01010101), co powoduje zapalanie parzystych diod LED. Następnie w nieskończonej pętli `Do…Loop` wszystkie bity w rejestrze portu D są przesuwane o jedną pozycje w lewo instrukcją `Rotate`. W instrukcji `Rotate` oprócz numeru portu, którego dotyczy, należy podać jeszcze parametr, w którą stronę będą przesuwane bity rejestru portu: w lewo lub w prawo (`Left, Right`). Instrukcja `Rotate` ma po przecinku parametr, który określa liczbę przesunięć podczas wykonywania tej instrukcji. Instrukcja `Waitms 200` wprowadza opóźnienie 200 ms w wykonywaniu instrukcji Rotate. Bez tego opóźnienia instrukcja `Rotate` byłaby wykonywana tak szybko, że efekt przesuwania świecenia diod nie byłby zauważalny. Kolejna instrukcja `Loop` kończy pętlę. Bez płytki testowej działanie programu można sprawdzić w programowym symulatorze, wykorzystując emulator sprzętowy z wirtualnymi diodami LED.

Przykład obsługi układu czasowego

Każdy mikrokontroler jest wyposażony w co najmniej jeden układ czasowy [1, 2, 10, 11, 12, 20, 21, 31, 35, 38], który można zazwyczaj skonfigurować do pracy w trybach: licznika, czasomierza czy generatora PWM. Dużą zaletą timerów jest to, że mogą pracować niezależnie od innych bloków funkcjonalnych mikrokontrolera - mikrokontroler pracuje w swoim rytmie a timer w swoim. W przedstawionym dalej przykładzie zastosowano Timer0 (pracujący w trybie czasomierza) do odmierzania dokładnie 1-sekundowych odcinków czasu, które mogą być wykorzystane do odmierzania dłuższych odcinków czasu, na przykład w zegarze czasu rzeczywistego. Do odmierzania tych odcinków czasu wykorzystano sygnał przerwania generowany po przepełnianiu licznika Timer0, które jest generowane co 8 ms. Dzięki wykorzystaniu przerwania od przepełnienia licznika, ten odcinek czasu jest dokładnie odmierzony i niezależny od instrukcji występujących w pętli głównej programu.

Przed użyciem timerów należy je wcześniej skonfigurować instrukcją `Config`. Dla Timer0 instrukcja konfigurująca go jako licznik jest następująca:

```
Config Timer0 = Counter, Prescale = 11 8 | 64 |
25611024, Edge = Rising|Falling, Clear Timer = 1|0
```

Natomiast skonfigurowanie Timer0 jako czasomierza następuje instrukcją:

```
Config Timer0 = Timer, Prescale = 1|8|64|256|1024
```

Timer0 w konfiguracji licznika jest często wykorzystywany do zliczania zewnętrznych impulsów, w tym przy pomiarach częstotliwości sygnałów. Natomiast Timer0 skonfigurowany jako czasomierz jest zazwyczaj wykorzystywany do odmierzania dokładnych odcinków czasu.

Gdy Timer0 jest skonfigurowany do pracy w charakterze licznika (parametr `Timer0 = Counter`), wtedy należy określić zbocze synchronizujące (parametr `Edge`). Ustala się więc czy zawartość licznika będzie zwiększana wraz z narastającym zboczem (`Rising`) czy opadającym (`Falling`) sygnału występującego na wejściu TO mikrokontrolera. Parametry `Prescale` oraz `Clear Timer` są opcjonalne i zależne od zastosowanego mikrokontrolera.

Gdy Timer0 jest skonfigurowany do pracy w charakterze czasomierza (parametr `Timer0 = Timer`), wtedy należy określić parametr `Prescale`. Parametr ten umożliwia wybranie współczynnika podziału częstotliwości sygnału taktującego Timer0, czyli zmniejszenie częstotliwości sygnału powodującego zmianę stanu licznika Timer0, pochodzącego z oscylatora mikrokontrolera. Częstotliwość sygnału z tego oscylatora, zanim trafi on na wejście Timer0 można podzielić w preskalerze, którego stopień podziału może być wybrany spośród wartości: 1, 8, 64, 256 oraz 1024. Jeśli w programie użyto instrukcji Config Timer0, tryb pracy licznika jest zapamiętywany przez kompilator oraz ustawiany jest rejestr sprzętowy TCCRO.

Licznik Timer0 jest wykorzystywany także przez niektóre instrukcje Bascoma. Może on być w dowolnej chwili zatrzymany instrukcją `Stop Timer0`. Jego uruchomienie umożliwia instrukcja `Start Timer0`. Bascom ma jedną instrukcję oraz zmienną `Counterx` (gdzie `x` to numer timera), która umożliwia wpisanie wartości początkowej do rejestru licznika (rejestru, w którym odbywa się zliczanie). Wpisywanie wartości początkowej do licznika umożliwia odmierzanie różnych odcinków czasowych. Składnia instrukcji wpisującej wartości do rejestru liczącego wybranego timera jest następująca:

```
Load  x, wartosc
```

gdzie:

`x` - nazwa licznika - może być: Timer0, Timerl lub Timer2,

`wartosc` - wartość określająca liczbę impulsów do zliczenia przez timer.

Instrukcja ta jest często wykorzystywana do odmierzania potrzebnych odcinków czasu, gdyż licznik Timer0 nie ma trybu zliczania z funkcją automatycznego przeładowywania zawartości. Zliczenie określonej instrukcją `Load` liczby impulsów będzie powodować przepełnienie timera, co umożliwia generowanie sygnału przerwania. Przed załadowaniem wartości początkowej do timera następuje jej obliczenie przez wykonanie działania: 256-`wartosc` w instrukcji `Load` (dla licznika 8-bitowego Timer0).

Dla obsługi timerów mikrokontrolera są w Bascomie dostępne dodatkowe zmienne (są one innymi nazwami ich rejestrów), które umożliwiają modyfikację oraz odczyt rejestrów timerów. Składnia takiej zmiennej dla modyfikacji wartości rejestru liczącego timera jest następująca:

```
Counterx   =   wartosc1
```

Natomiast przy odczycie stanu rejestru liczącego timera należy ją stosować następująco:

```
wartosc2   =  Counterx
```

gdzie:

`wartosc1` - zmienna typu Byte, Integer lub Word, albo stała liczbowa określająca nową zawartość licznika;

`wartosc2` - zmienna, do której jest przekazywana aktualna zawartość timera;

`x` - numer timera (licząc od 0).

Aby odmierzyć za pomocą Timer0 jedną sekundę, przy częstotliwości rezonatora kwarcowego 8 MHz, należy przeprowadzić obliczenia optymalnego współczynnika podziału preskalera oraz wartości wpisywanej do licznika Timer0. Przy częstotliwości taktowania 8 MHz, jeden okres sygnału zegarowego wynosi 125 ns. Aby odmierzyć wartość 1 sekundy należy 8 MHz podzielić tak, aby uzyskać sygnał o częstotliwości 1 Hz, czyli o okresie 1 sekundy. Przy podziale preskalera przez 256 otrzyma się okres impulsów zliczanych przez Timer0 równy 32 µs. Zliczenie w liczniku 250 tych impulsów da wartość 8 ms. Licznik Timer0 będzie się przepełniał co 8 ms, więc sygnał przerwania będzie generowany co

8 ms. Kolejne zliczenie 125 odcinków po 8 ms (czyli liczby generowanych przerwań) da właśnie 1 sekundę. Podsumujmy, częstotliwość 8 MHz jest dzielona przez następujące wartości 8 MHz/ 256/250/125 = 1, oczywiście przy innych częstotliwościach oscylatora należy obliczyć odpowiednie wartości współczynników podziału tej częstotliwości. Jeżeli timer ma odmierzać precyzyjnie odcinki czasu (jak np. w zegarach, stoperach), to do taktowania mikrokontrolera należy stosować oscylatory z rezonatorami kwarcowymi, gdyż gwarantują one dobrą stabilność częstotliwości generowanego sygnału. Do tego celu na pewno nie nadaje się wewnętrzny oscylator RC mikrokontrolera.

Aby sprawdzić ten sposób odmierzania 1 sekundy, można wykorzystać schemat dołączenia diody LED do linii PDO przedstawionego na rysunku 5.2. Po odmierzeniu 1 sekundy można w podprogramie obsługi przerwania zmienić na przeciwny stan linii portu, do której dołączono diodę LED. Wówczas dioda LED będzie załączana oraz wyłączana na czas 1 sekundy.

Przykładowy program, w którym wykorzystano Timer0 i jego przerwania do odmierzania 1 sekundy przedstawiono na wydruku 5.2.

Wydruk 5.2. Przykład programu z wykorzystaniem Timer0 do odmierzania 1 sekundy:

```
'Przyklad wykorzystania Timer0 do generowania
'w przerwaniu 1 sekundowych odcinków czasu
'Program co sekundę w przerwaniu zmienia na przeciwny
'stan linii PDO sterującej diodą LED
'Przerwanie od przepełnienia Timer0 jest generowane
'co 3 ms
'(8MHz/256/250 = 125, a 1/125 s - 8 ms)
'Zliczenie 125 odcinków 8 ms da 1 sekundę
$REGFILE = "m8def.dat"      'informuje kompilator o
                            'pliku dyrektyw
                            'mikrokontrolera
$CRYSTAL = 8000000          'informuje kompilator o
                            'częstotliwości oscylatora
                            'taktującego
                            'mikrokontroler
Config Pind.0 = Output      'linia PDO jako wyjściowa
Config Timer0 = Timer ,Prescale = 256
                'konfiguracja Timer0 jako timera
                'z podziałem preskalera przez 256
On Timer0 Odmierz_ls  'przerwanie od przepełnienia
                      'Timer0 o etykiecie Odmierz_ls
Dim Licz_8ms As Byte  'zmienna pomocnicza zliczająca
```

```
                        'odcinki czasu równe 8 ms
Enable Interrupts       'odblokowanie globalnego
                        'systemu przerwań

Enable Timer0           'odblokowanie przerwania od
                        'przepełnienia Timer0
Load Timer0 = 250       'wartość początkowa Timer0
Do                      'pętla nieskończona
Loop
End                     'koniec programu

Odmierz 1s:             'początek podprogramu obsługi
                        'przerwania od przepełnienia
                        'Timer0
  Load Timer0 = 250     'wartość początkowa Timer0
  Incr Licz_8ms         'zwiększ o jeden wartość
                        'zmiennej pomocniczej Licz_8ms
  If Licz_8ms = 125 Then    'jeżeli wartość tej
                            'zmiennej
                            'równa 125 (125*8 ms = 1)
                            'to 'odliczono 1 sekundę
   Licz_8ms = 0         'zerowanie zmiennej licznikowej
   Toggle Portd.0       'zmień na przeciwny stan linii
                        'PDO portu D mikrokontrolera
  End If                'koniec warunku If...Then
Return                  'powrót z przerwania
```

Najpierw linia PD0 portu D (sterująca diodą LED) jest konfigurowana jako wyjściowa. Następna instrukcja `Config` konfiguruje Timer0 jako czasomierz z podziałem preskalera 256. W kolejnej instrukcji On jest określany rodzaj oraz nazwa podprogramu, do którego nastąpi skok podczas wystąpienia sygnału wybranego przerwania. W tej instrukcji źródłem przerwania jest Timer0 (w momencie jego przepełnienia), po czym następuje skok podczas jego obsługi do podprogramu z etykietą *Odmierz_1s*. Zmienna *Licz_8ms* służy do zliczania 8-mio ms odcinków czasu, których musi być 125 dla uzyskania 1 sekundy. Następnie odblokowywany zostaje globalny system przerwań oraz przerwanie od przepełnienia Timer0, a do rejestru liczącego licznika jest wpisywana wartość impulsów do odmierzenia, w tym przypadku 250 impulsów, czyli co 8 ms.

Aby Timer0 odmierzał 8 ms, to w podprogramie obsługi przerwania jako pierwsza powinna wystąpić instrukcja ładująca do rejestru liczącego licznika wartość 250 impulsów do odliczenia, gdyż dla tego timera nie jest dostępna funkcja automatycznego przeładowywania zawartości. Dlatego należy zadbać o załadowanie do timera potrzebnej wartości

początkowej. Następnie w programie jest nieskończona pętla `Do…Loop`. Po instrukcji `End` w programie, zaczyna się podprogram obsługi przerwania zgłaszanego co 8 ms. Aby odmierzyć czas 1 sekundy należy zliczyć 125 przerwań. Do ich zliczania zastosowano zmienną *Licz_8ms*, która zwiększana o jeden (przez instrukcję `Incr`) po każdym sygnale przerwania. Jeżeli wartość tej zmiennej osiągnie 125, to zostaną wykonane instrukcje zawarte w instrukcji warunkowej `If…Then`. Gdy spełniony jest warunek w tej instrukcji, zmienna *Licz_8ms* zostaje wyzerowana, co przygotowuje ją do ponownego odliczenia 1 sekundy oraz zmieniony zostanie na przeciwny stan linii PD0 portu D mikrokontrolera. Podprogram przerwania kończy instrukcja `Return`.

Ponieważ licznik po zgłoszeniu i wykonywaniu podprogramu przerwania nadal pracuje, to przed wykonaniem instrukcji `Load Timer0 = 250` (ładującej wartość początkową) zliczy już pewną liczbę impulsów, które pogarszają dokładność odmierzanego czasu. Dokładność odmierzanego czasu można zwiększyć wykorzystując do załadowania wartości początkowej timera zapis: `Timer0 = Timer0 + 6` (bo 256 - 250 = 6). Zapis w takiej postaci wartości początkowej timera kompensuje upływ czasu od wykrycia zgłoszenia sygnału przerwania do rozpoczęcia jego wykonywania.

Przykład obsługi wyświetlacza LCD

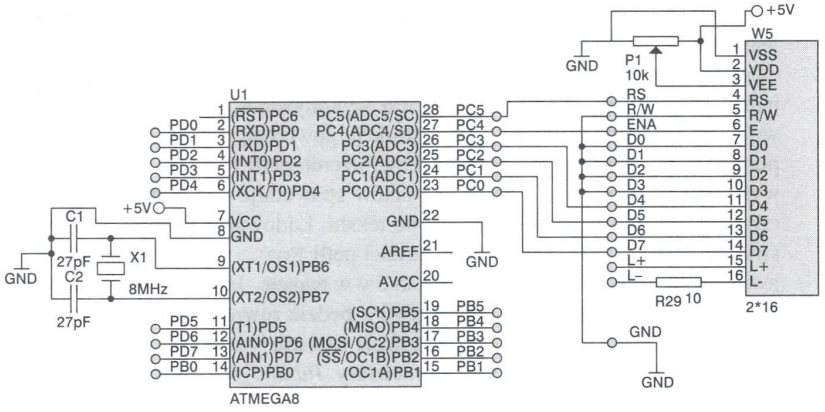
Sterowanie wyświetlaczami alfanumerycznymi LCD ze sterownikiem HD44780 jest w języku Bascom stosunkowo łatwe [19, 45]. Do ich obsługi przewidziano wiele instrukcji, których użycie zostanie zilustrowane w tym przykładzie. Możliwe jest podłączenia wyświetlacza alfanumerycznego LCD do mikrokontrolera na dwa sposoby:

- Łącząc bezpośrednio wyprowadzenia wyświetlacza LCD z wyprowadzeniami portów mikrokontrolera. Tryb ten jest nazywany *Pin Mode*. Projektant ma pełną swobodę dołączania wyprowadzeń LCD do dowolnych portów mikrokontrolera. Można w ten sposób zoptymalizować mozaikę połączeń na płytce drukowanej, lecz okupione jest to zwiększeniem rozmiaru kodu potrzebnego do sterowania wyświetlaczem.

- · Łącząc wyprowadzenia linii danych wyświetlacza do systemowej szyny danych, która jest dostępna w systemach mających możliwość współpracy z zewnętrzną pamięcią danych XRAM. Jest to tzw. tryb *Bus Mode*. Przy tym podłą-

czeniu wyświetlacz może pracować w trybie 4-bitowym lub 8-bitowym.

W prezentowanym przykładzie, wyświetlacz LCD o organizacji 2x16 znaków bezpośrednio dołączono do wyprowadzeń mikrokontrolera w trybie pracy 4-bitowej. Ten tryb pracy jest często stosowany, gdyż oszczędza się 4 linie mikrokontrolera. Do sterowania wyświetlaczami w trybie 4-bitowym potrzeba jedynie 6 linii mikrokontrolera. Na rysunku 5.3 przedstawiono schemat dołączenia wyświetlacza alfanumerycznego LCD do mikrokontrolera.



Rysunek 5.3. Schemat dołączenia wyświetlacza LCD do mikrokontrolera

Poszczególne wyprowadzenia wyświetlacza alfanumerycznego pełnią następujące funkcje:

- VSS – minus zasilania;
- VDD – plus zasilania;
- VEE – regulacja kontrastu;
- RS – wybór rejestru;
- R/W – zapis/odczyt rejestrów wyświetlacza;
- E – sygnał zezwalający;
- D0…D7 – linie danych;
- L+ – anoda diod podświetlających;
- L- – katoda diod podświetlających.

Do skonfigurowania wyświetlacza do pracy w trybie 4-bitowym, dołączonego wprost do linii portu mikrokontrolera wystarczą dwie instrukcje konfigurujące: `Config Lcd` oraz `Config Lcdpin`. Instrukcja `Config Lcd` umożliwia wybór rodzaju wyświetlacza. Do wyboru są wyświetlacze o następującej organizacji znaków: 4x40, 1x16, 2x16, 4x16, 2x20 i wiele innych. Instrukcja `Config Lcdpin` umożliwia przypisanie linii portu mikrokontrolera do odpowiednich wejść wyświetlacza LCD. Innymi instrukcjami służącymi do sterowania wyświetlaczem są: instrukcja `Cls`, która czyści ekran wyświetlacza (kasuje wyświetlane znaki). Po jej zastosowaniu kursor jest ustawiany na pierwszej pozycji w pierwszym wierszu wyświetlacza. Kolejna instrukcja to `Lcd`, która spowoduje wyświetlenie łańcucha znaków podanych w cudzysłowie. Zastosowanie instrukcji `Lowerline` ustawia kursor na początku drugiej linii wyświetlacza. Dostępne są także instrukcje `Upperline` - ustawia kursor na początku pierwszej linii, `Thirdline` - ustawia kursor na początku w trzeciej linii i `Fourthline`, która ustawia kursor na początku czwartej linii. Oczywiście wykorzystanie tych instrukcji będzie zależeć od liczby linii, jakie ma zastosowany wyświetlacz. Do ustawienia kursora na określonej parametrami pozycji służy instrukcja `Locate`, której składnia jest następująca:

```
Locate y , x
```

gdzie:

`x` - pozycja kursora w linii o zakresie wartości od 1 do 64 - zależnie od używanego wyświetlacza,

`y` - numer linii wyświetlacza o zakresie od 1 do 4 - zależnie od rodzaju używanego wyświetlacza.

Na wydruku 5.3 przedstawiono program obsługi wyświetlacza LCD z wykorzystaniem wyżej omówionych instrukcji.

```
'Program sterujący wyświetlaczem alfanumerycznym LCD
'2*16 znaków w trybie pracy 4-bitowej
$REGFILE = "m8def.dat"      'informuje   kompilator
                            'o pliku dyrektyw
                            'mikrokontrolera
$CRYSTAL = 8000000    'informuje kompilator
                      'o częstotliwości oscylatora
                      'taktującego mikrokontroler
Config Lcd = 16 * 2   'konfiguracja typu LCD
```

```
Config Lcdpin = Pin , Db4 = Portc.3 , Db5 = Portc.2 ,
Db6 = Portc.1 , Db7 = Portc.0 , E = Portc.4 ,
Rs = Portc.5
                        'konfiguracja linii, do których
                        'dołączono wyświetlacz
Dim I As Byte           'zmienna licznikowa


Cls                     'czyści LCD
Lcd "*Programowanie*" 'wyświetlenie w pierwszej linii
                        'tekstu *Programowanie*
Wait 1                  'opóźnienie 1 sekundy
Lowerline               'wybranie drugiej linii
Wait 1                  'czekaj 1 sekundę
Lcd "  Przesuwaj  "    'wyświetlenie w drugiej linii
                        'tekstu Przesuwaj
Wait 1                  'opóźnienie 1 sekundy
For I = 1 To 10         'w pętli tekst zostanie
                        'przesunięty o 10 pozycji
Shiftlcd Right          'w prawo dla wszystkich linii
                        'wyświetlacza
Waitms 500              'opóźnienie przesunięć o 500 ms
Next I                  'zwiększenie I o jeden
For I = 1 To 10         'w pętli tekst zostanie
                        'przesunięty o 10 pozycji
Shiftlcd Left           'w lewo dla wszystkich linii
                        'wyświetlacza
Waitms 500              'opóźnienie przesunięć o 500 ms
Next I                  'zwiększenie I o jeden
Locate 2 , 2            'ustawienie pozycji kursora na
                        'drugą linię i drugą pozycję
Lcd "*"                 'wyświetlenie na wybranej
                        'pozycji kursora znaku *
Wait 1                  'opóźnienie 1 sekundy
Shiftcursor Right       'przesunięcie kursora o jedną
                        'pozycję w prawo
Lcd "%"                 'wyświetlenie znaku % na
                        'przesuniętej pozycji kursora
Wait 1                  'opóźnienie 1 sekundy
Home Upper              'wybranie pierwszej linii
                        'i powrót kursora na jej
                        'początek
Led "Inny tekst"        'zastąpienie tekstu w pierwszej
                        'linii tekstem Inny tekst
End                     'koniec programu
```

W programie dokonano wyboru wyświetlacza a następnie za pomocą instrukcji `Config Lcdpin` przypisano odpowiednie linie portów

mikrokontrolera do odpowiednich wejść wyświetlacza zgodnie ze schematem przedstawionym na rysunku 5.3.

Zdefiniowana w programie zmienna I będzie pomocna do przedstawienia działania wyświetlacza LCD z różnymi instrukcjami. Analizując program natrafiamy na instrukcje `Cls`, która czyści ekran wyświetlacza (kasuje wyświetlane znaki). Po jej zastosowaniu kursor jest ustawiany na pierwszej pozycji w pierwszym wierszu wyświetlacza. Kolejna instrukcja `Lcd` spowoduje wyświetlenie łańcucha znaków, który podano w cudzysłowie. Na wyświetlaczu w pierwszej linii zostanie wyświetlony tekst **Programowanie**.

Po odczekaniu sekundy wykonana zostanie instrukcja `Lowerline`, która ustawia kursor na początku drugiej linii wyświetlacza.

Wykonanie następnej instrukcji `Lcd` wyświetli w drugiej linii wyświetlacza tekst *Przesuwaj*. Następnie w pętli `For…Next` zostanie wykonana 10 razy instrukcja `Shiftlcd` z parametrem `Right`, powodująca przesuwanie tekstu na wyświetlaczu LCD o jedną pozycję w prawo. W kolejnej pętli `For…Next` ponownie zostanie wykonana 10 razy instrukcja `Shiftlcd`, ale z parametrem `Left`, która spowoduje przesuwanie tekstu na LCD w lewo. Wpisany wcześniej tekst powróci więc na swoją pierwotną pozycję. Opóźnienie 500 ms wprowadzono w tych pętlach, aby przesuwanie tekstu było widoczne. Po wykonaniu dwóch pętli `For…Next` jest wykonywana instrukcja `Locate`, która spowoduje ustawienie kursora na określonej parametrami pozycji.

Po wykonaniu tej instrukcji w programie, kursor będzie ustawiony w drugiej linii i na pozycji drugiego znaku. Na tej pozycji, po wykonaniu następnej instrukcji, zostanie wyświetlony znak *. Po odczekaniu 1 sekundy jest wykonywana instrukcja `Shiftcursor` z parametrem `Right`. Powoduje ona przesuwanie kursora wyświetlacza LCD o jedną pozycję w prawo lub lewo, w zależności od jej parametru. W użytej instrukcji kursor zostanie przesunięty o jedną pozycję w prawo. Na pozycji kursora zostanie wyświetlony znak %. Będzie on wyświetlony na pozycji 4-tego znaku wyświetlacza, gdyż wykonanie instrukcji `Lcd` przesuwa kursor na następną pozycję, na której ma być wyświetlony następny znak. Kursor został wcześniej przesunięty z pozycji 2 na pozycje 3 po wykonaniu instrukcji `Lcd "*"`. Potem została wykonana instrukcja `Shiftcursor`, która spowodowała także przesunięcie kursora, dlatego znak % zostanie wyświetlony na 4 pozycji.

Po odczekaniu także 1 sekundy zostanie wykonana instrukcja `Home` z opcjonalnym parametrem `Upper`. W przykładzie kursor ustawiono na początku pierwszej linii wyświetlacza.

Następnie do tej linii zostaje zapisany tekst *Inny tekst,* który nadpisze poprzednio wpisany tekst **Programowanie**.

Przetestowanie działania tego programu pozwoli zapoznać się z obsługą programową wyświetlaczy LCD, gdyż ilustruje on działanie wielu instrukcji przeznaczonych do obsługi podobnych urządzeń peryferyjnych. Opisany program można przetestować bez układu uruchomieniowego w symulatorze programowym, wykorzystując wirtualny wyświetlacz LCD. Przy symulacji programowej należy na początku programu umieścić dyrektywę $SIM, która spowoduje, że kompilator opuści instrukcje opóźnień, które bardzo spowolnią pracę symulatora.

Przykład obsługi przycisków

Ważnymi elementami towarzyszącymi mikrokontrolerom, są oprócz wyświetlaczy przyciski oraz klawiatury [7, 45]. Za pomocą tych elementów jest możliwe wprowadzanie do mikrokontrolera odpowiednich wartości lub przełączanie jego trybów pracy. Za ich pomocą operator może wprowadzać dane lub modyfikować wartości konfigurowanych parametrów programu. Przyciski do mikrokontrolera można dołączać bezpośrednio do jego wyprowadzeń lub dołączyć je matrycowo.

Ponieważ styki klawiatury są elementami mechanicznymi, ich obsługa może się wydawać trudna i sprawiać trochę problemów. Problemy te wynikają z faktu, że po naciśnięciu przycisku jego styki przez pewien czas drgają powodując powstanie serii impulsów zamiast „czystego" zwarcia lub rozwarcia. Drgania styków zazwyczaj nie trwają dłużej niż 25 ms. Jeżeli mikrokontroler będzie bardzo często sprawdzał stan linii, do której dołączono przycisk, to zamiast pojedynczego naciśnięcia może wykryć serię naciśnięć, co może być przyczyną niepoprawnej pracy programu. Najczęściej stosowany i najprostszy sposób rozwiązania tego problemu polega na odczytywaniu przycisku (klawiatury) z opóźnieniem. Po wykryciu naciśnięcia przycisku odmierzane jest zazwyczaj opóźnienie ok. 25 ms, po czym ponownie jest sprawdzany stan przycisku, który można już uznać za stabilny, jeżeli jest dalej naciśnięty.

W języku Bascom występuje instrukcja `Debounce` umożliwiająca wykrycie naciśnięcia przycisku(ów) na wybranych liniach portów mikrokontrolera. Zapewnia ona odczytanie stanu przycisku z zastosowaniem

programowej eliminacji drgań styków. Składnia tej instrukcji jest następująca:

```
Debounce Pinx.y, stan, etykieta [, Sub]
```

gdzie:

`Pinx. y` - linia portu, która będzie sprawdzona, np. Pinb.5;

`stan` - 0, gdy sygnałem aktywnym naciśnięcia ma być poziom niski, a 1 gdy poziom wysoki;

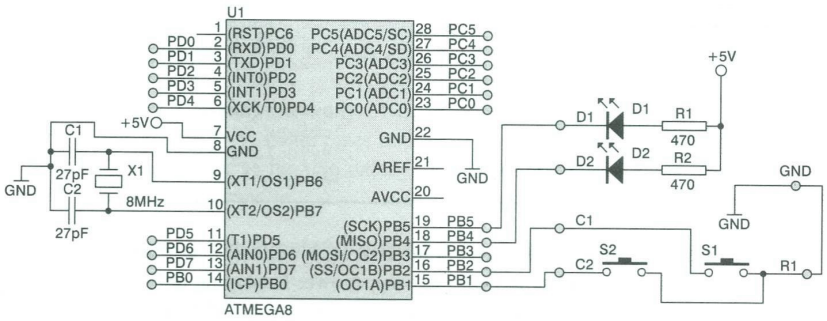
`etykieta` - etykieta określająca miejsce skoku po naciśnięciu przycisku;

`Sub` - po naciśnięciu nastąpi skok do podprogramu o podanej etykiecie.

Parametr `Sub` umożliwia skok do etykiety podprogramu, z którego będzie możliwy powrót instrukcją `Return`. Instrukcja `Debounce` po wykryciu zmiany stanu linii portu do stanu określonego parametrem `stan`, po 25 ms sprawdza ponownie stan tej linii (eliminuje to drganie styków). Jeśli stan się nie zmienił, to następuje skok do podanej etykiety. Jeżeli będą różne stany (parametr `stan` będzie różny od stanu linii portu), to będą wykonane kolejne instrukcje programu zawarte po tej instrukcji. Instrukcja ta nie wstrzymuje bowiem działania programu. Przy przytrzymaniu przycisku instrukcja `Debounce` zostanie wykonana tylko jeden raz. Jej ponowne wykonanie będzie możliwe po puszczeniu oraz ponownym naciśnięciu przycisku.

Każda instrukcja `Debounce` korzysta z 1 bitu pamięci RAM, do zapamiętania stanu linii portu. Dla każdej linii portu jest przydzielany osobny bit.

Jeżeli w programie jest potrzebne oczekiwanie na naciśnięcie przycisku, to można skorzystać z instrukcji `Bitwait`, która wstrzymuje działanie programu dopóty, dopóki na określonej linii portu pojawi się oczekiwany stan (np. przy naciśnięciu przycisku). Dla instrukcji `Debounce` jest możliwa zmiana czasu opóźnienia (domyślnie 25 ms) na inny. Służy do tego instrukcja konfigurująca `Config Debounce = czas`, gdzie `czas` jest wartością w ms. Aby sprawdzić działanie oraz wykorzystanie instrukcji `Debounce`, do mikrokontrolera można dołączyć dwa przyciski oraz dwie diody LED zgodnie ze schematem na rysunku 5.4 i przygotować program z tą instrukcją.



Rysunek 5.4. Schemat dołączenia do mikrokontrolera przycisków oraz diod LED

Przyciski mogą być dołączone do linii PB1 oraz PB2, a diody do PB4 oraz PB5 mikrokontrolera. Przyciśnięcie przycisku będzie powodować podanie na linię wejściową niskiego poziomu napięcia. Tak więc instrukcja `Debounce` powinna „reagować" na poziom niski na danej linii. Napisany przykładowy program dla mikrokontrolera będzie powodował, że po naciśnięciu przycisku S1 będzie zmieniał się na przeciwny stan diody D1, a naciśnięcie przycisku S2 będzie powodowało zmianę stanu diody D2 na przeciwny. Na wydruku 5.4 przedstawiono przykład programu odczytu stanu dwóch przycisków za pomocą instrukcji `Debounce`.

Wydruk 5.4. Program odczytu stanu dwóch przycisków za pomocą instrukcji Debounce:

```
'Program obsługi przycisków S1 i S2 za pomocą
'instrukcji Debounce
$REGFILE = "m8def.dat" 'informuje kompilator o pliku
                       'dyrektyw mikrokontrolera
$CRYSTAL = 8000000     'informuje kompilator
                       'o częstotliwości oscylatora
                       'taktującego mikrokontroler
Config Pinb.1 = Input 'linia PB1 jako wejściowa
Config Pinb.2 = Input 'linia PB2 jako wejściowa
Config Pinb.4 = Output      'linia PB4 jako wyjściowa
Config Pinb.5 = Output      'linia PB5 jako wyjściowa

Led1 Alias Portb.5    'przypisanie nazwie Portb.5
                      'nazwy Led1
Led2 Alias Portb.4    'przypisania nazwie Portb.4
                      'nazwy Led2
S1 Alias Pinb.2       'przypisanie nazwie Pinb.2
                      'nazwy S1
```

```
S2 Alias Pinb.l       'przypisanie nazwie Pinb.l
                  'nazwy S2
Set Portb.l      'dołączenie do linii PB1
                 'rezystora podciągającego
Set Portb.2      'dołączenie do linii PB2
                 'rezystora podciągającego
Do               'nieskończona pętla Do...Loop
Debounce S1,0,Pr1,Sub 'jeśli naciśnięty
                      'przycisk S1 to skok
                      'do podprogramu Pr1
Debounce S2,0,Pr2,Sub 'jeśli naciśnięty
                      'przycisk S2, to skok
                      'do podprogramu Pr2
Loop End              'koniec programu

Pr1:                  'podprogram Pr1
  Toggle Led1         'zmiana na przeciwny stanu
                      'wyjścia sterującego diodą D1
Return                'powrót z podprogramu

Pr2:                  'podprogram Pr2
  Toggle Led2         'zmiana na przeciwny stanu
                      'wyjścia sterującego dioda D2
Return                'powrót z podprogramu
```

W pierwszej kolejności w programie linie PB1 oraz PB2 są konfigurowane jako wejściowe, a linie PB4 oraz PB5 jako wyjściowe. Następnie do wykorzystywanych linii portów przypisano aliasy, które oznaczają elementy, jakie do tych linii dołączono. Instrukcje `Set Portb.l` oraz `Set Portb.2` dołączają do linii PB1 oraz PB2 rezystory podciągające. Należy takie rezystory dołączyć, gdyż przy rozwartych przyciskach byłby na tych liniach stan nieokreślony, a rezystory podciągające wymuszają na nich poziomy wysokie.

Dalej, w nieskończonej pętli `Do…Loop` umieszczono dwie instrukcje `Debounce` odczytujące odpowiednio stan przycisku S1 oraz S2. Po naciśnięciu przycisku, na linii wejściowej będzie poziom niski, dlatego parametr `stan` instrukcji `Debounce` ma wartość 0. Po wykryciu przez te instrukcje poziomu niskiego nastąpią skoki do podprogramów oznaczonych etykietę *Pr1* dla S1, oraz *Pr2* dla S2, gdyż w instrukcjach `Debounce` zastosowano parametr `Sub`. Po naciśnięciu przycisku S1 wykonane zostaną instrukcje w podprogramie *Prl,* czyli zostanie zmieniony na przeciwny stan linii PB5 (Led1). Po naciśnięciu przycisku S2 wywołany zostanie podprogram *Prl* i zmieniony zostanie na przeciwny stan linii PB4 (Led2). Przytrzymanie wciśniętego przycisku nie powoduje żad-

nych zmian. Dopiero po jego puszczeniu i ponownym naciśnięciu wykonane zostaną od początku odpowiednie podprogramy.

Przykład obsługi przetwornika A/C

Wiele mikrokontrolerów AVR jest wyposażonych w wewnętrzny przetwornik A/C, 8-bitowy lub 10-bitowy [1, 2, 11]. Wewnętrznym przetwornikiem w mikrokontrolerze ATmega8 można w danej chwili odczytywać napięcie tylko z jednej linii ADC0...ADC5, dołączanej do przetwornika poprzez multiplekser. Po odczycie napięcia z danej linii można więc przełączyć multiplekser na kolejną linię.

Jako napięcie odniesienia dla tego przetwornika może być stosowane napięcie zasilania 5V, napięcie z wewnętrznego źródła odniesienia 2,56V oraz napięcie zewnętrzne podane na wejście AREF mikrokontrolera. Przetwornik jest zasilany dobrze filtrowanym napięciem podanym na linie AGND i AVCC mikrokontrolera. Spośród 6 linii wejściowych przetwornika, tylko z 4 linii (ADCO... ADC3) napięcie wejściowe jest przetwarzane z rozdzielczością 10 bitów, natomiast napięcie podawane na linie ADC4 i ADC5 jest przetwarzane z rozdzielczością 8 bitów.

W języku Bascom opracowano do obsługi wewnętrznego przetwornika instrukcję konfigurującą oraz funkcję umożliwiającą odczyt cyfrowej wartości przetworzonego napięcia z wybranego wejścia ADCx. Instrukcja konfigurująca `Config Adc` określa sposób pracy wbudowanego przetwornika. Składnia tej instrukcji jest następująca:

```
Config Adc = Single|Free, Prescaler = dzielnik|Auto,
Reference = Off|Avcc|Internal
```

gdzie:

`Adc` - określa tryb pracy, który może być `Single` lub `Free`;

`Prescaler` - określa stopień podziału sygnału częstotliwości zegara systemowego - dozwolone są wartości: `2, 4, 8, 16, 32, 64` lub `128`, a ustawienie Auto spowoduje, że kompilator dostosuje wartość dzielnika do bieżącej częstotliwości sygnału taktowania;

`Reference` - niektóre mikrokontrolery (w tym ATmega8) mają dodatkowe opcje dotyczące napięcia odniesienia - można ustawić: `Off, Avcc` lub `Internal`.

Ustawienie preskalera w tryb `Auto` spowoduje, że dzielnik zostanie tak dobrany, aby częstotliwość taktowania przetwornika była najwyższa

z możliwych dla danej częstotliwości sygnału taktującego mikrokontroler.

Dla parametru `Reference` można przyjąć następujące opcje:

- `Off` - napięcie odniesienia jest pobierane z wyprowadzenia AREF mikrokontrolera, przy czym wewnętrzne źródło napięcia odniesienia jest odłączone;
- `Avcc` - napięcie odniesienia jest pobierane z wyprowadzenia AVCC mikrokontrolera, przy czym do wyprowadzenia AREF należy dołączyć kondensator;
- `Internal` - napięcie odniesienia jest pobierane z wewnętrznego źródła 2,56 V, przy czym do wyprowadzenia AREF należy dołączyć kondensator.

Do odczytania wartości z przetwornika służy funkcja `Getadc`. Wywołanie tej funkcji jest następujące:

```
zmienna = Getadc(nr_kanalu)
```

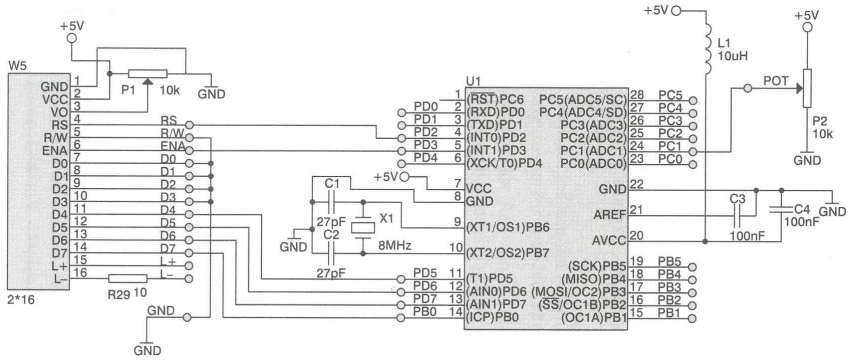
gdzie:

`zmienna` - zmienna, której jest przypisywana odczytana z przetwornika wartość napięcia;

`nr_kanału` - numer kanału przetwornika A/C, z którego jest odczytywana wartość - może być z zakresu 0...7.

Funkcja `Getadc` może być stosowana jedynie dla mikrokontrolerów z wbudowanym przetwornikiem A/C, skonfigurowanym w trybie `Single`. Wyprowadzenia wejściowe przetwornika A/C mogą być także wykorzystywane jako normalne linie portów. Należy jednak pamiętać, aby podczas pracy przetwornika ich stan nie był zmieniany. Aby można było korzystać z przetwornika A/C, należy go wcześniej włączyć instrukcją `Start Adc`.

W przykładzie zilustrowano sposób odczytywania jednej wartości napięcia podanego na jedno z wejść wbudowanego przetwornika A/C. Wyniki, to jest odczytana z przetwornika wartość liczbowa i po przeliczeniu jej na wartość napięcia, są wyświetlane na wyświetlaczu LCD. Na rysunku 5.5 przestawiono schemat, w którym do wejścia ADC1 przetwornika dołączono potencjometr, dzięki któremu jest możliwe podanie na przetwornik napięcia z zakresu 0 do 5V.



Rysunek 5.5. Schemat wykorzystania wewnętrznego przetwornika A/C mikrokontrolera do pomiaru napięcia z potencjometru

Napięcie podane na linię ADC1 jest przetwarzane z rozdzielczością 10 bitów, czyli przetwornik może odczytać napięcie wejściowe (przy napięciu odniesienia 5V) z rozdzielczością 5V /1024 = ok. 4,88 mV. Napięcie zasilające przetwornik jest filtrowane przez elementy C4, L1.

Na wydruku 5.5 przedstawiono program obsługi wewnętrznego przetwornika A/C wbudowanego w mikrokontroler ATmega8.

Wydruk 5.5. Program obsługi wbudowanego w mikrokontroler przetwornika A/C:

```
'Program obsługi wbudowanego w mikrokontroler
'10-bitowego przetwornika A/C
$REGFILE = "m8def.dat"      'informuje kompilator o
                            'pliku dyrektyw
                            'mikrokontrolera
$CRYSTAL = 8000000     'informuje kompilator
                       'o częstotliwości oscylatora
                       'taktującego mikrokontroler
Config Lcd = 16 * 2    'konfiguracja typu wyświetlacza
                       'LCD
Config Lcdpin = Pin , Db4 = Portd.5 , Db5 = Portd.6 ,
Db6 =   Portd.7 , Db7 = Portb.O , E = Portd.3 , Rs =
Portd.2
                       'konfiguracja linii
                       'mikrokontrolera, do których
                       'dołączono wyświetlacz LCD
Config Adc = Single , Prescaler = Auto , Reference =
Avcc
                       'konfiguracja wewnętrznego
                       'przetwornika A/C
Dim Wart_ac As Word    'zmienna na wartość odczytaną
```

```
                        'z przetwornika
Dim V As Single         'zmienna przechowując obliczoną
                        'wartość zmierzonego napięcia
Dim Wart_nap As String * 3  'zmienna, do której
                            'wpisywana
                        'zostaje przetworzona wartość
                        'zmiennej V na tekst
Start Adc       'uruchamia wbudowany przetwornik
Do              'początek pętli programu
Wart_ac = Getadc(1)   'odczytanie wartości z wejścia
                      'ADC1 mikrokontrolera
Cls                   'czyszczenie LCD
Lcd Wart_ac           'wyświetlenie w pierwszej linii
                      'Lcd odczytanej wartości
                      'z przetwornika
Lowerline             'kursor do drugiej linii LCD
V = Wart_ac * 0.0049  'zamiana odczytanej wartości
                      'z A/C na napięcie
                      'Wart_ac * ok. 4,8 mV
Wart_nap = Fusing(V,"#.&&") 'formatowanie
                            'wartości zmiennej
                      'V do do formatu x.xx i zamiana
                'tej wartości na postać tekstową
Lcd Wart_nap    'wyświetlenie obliczonej
                'wartości Wart_nap w drugiej
                'linii LCD
Waitms 100      'opóźnienie 100 ms
Loop            'koniec pętli Do...Loop
End             'koniec programu
```

Po skonfigurowaniu wyświetlacza LCD występuje instrukcja konfigurująca wewnętrzny przetwornik A/C. Aby można było zastosować do odczytywania funkcję `Getadc`, przetwornik skonfigurowano do pracy w trybie `Single`. Preskaler skonfigurowano z opcją `Auto`, napięcie referencyjne ustawiono z opcją `Avcc`, to znaczy będzie pobierane z linii AVCC zasilającej przetwornik, a więc będzie wynosiło 5 V.

Po instrukcji konfigurującej definiowane są zmienne. W zmiennej *Wart_ac* typu `Word` (gdyż wartość odczytana z przetwornika jest 10-bitowa) będzie pamiętana wartość odczytana z przetwornika. Pozostałe dwie zmienne są wykorzystywane podczas przeliczania wartości odczytanej z przetwornika na odpowiadającą jej wartość napięcia. Następnie instrukcją `Start Adc` jest inicjalizowany przetwornik. W nieskończonej pętli `Do…Loop` następuje odczyt wartości z przetwornika odpowiadającej wartości napięcia na wejściu ADC1 mi-

krokontrolera. Aby zmierzyć napięcie na wejściu ADC1, należy użyć funkcji `Getadc` z parametrem 1:

```
Wart_ac = Getadc(1) 'odczytuje wartość z wejścia ADC1
```

Po odczytaniu danych z przetwornika na wyświetlaczu LCD jest wyświetlana odczytana 10-bitowa liczba (w postaci dziesiętnej). Następnie odczytane z przetwornika dane są przeliczane na wartość napięcia, w taki sposób że liczba odczytana z przetwornika jest mnożona przez wartość 0,0049, gdyż 5/1024 ≈ 4,88 mV. Na przykład, odczytana z przetwornika wartość 823 odpowiada napięciu na jego wejściu 823 • 4,9 mV ≈ 4,0 V. Odczyt wartości z przetwornika odbywa się w nieskończonej pętli co 100 ms.

6 Literatura

1. *ATmega8, 8-bit AVR with 8K Bytes In-System Programmable Flash*, http://www.datasheetcatalog.org/datasheet/atmel/2486S.pdf

2. Baranowski R.: *Mikrokontrolery AVR ATmega w praktyce*, WBTC, Warszawa 2005

3. *BASCOM-AVR Demo version*: http://www.mcselec.com/index.php

4. *BASCOM-AVR help reference*: © MCS Electronics, 1995-2011

5. *BASCOM BASIC AVR Wersja 1.11.7.3*, Copyright by Zbigniew Gibek. Poland 2002-2003.

6. Bilski T.: *Interfejsy i urządzenia zewnętrzne*, Wydawnictwo Politechniki Poznańskiej, Poznań 2007

7. Bishop O.: *Electronics a first course*, Published by Elsevier Ltd., Oxford 2011

8. Bogusz J.: *Lokalne interfejsy szeregowe w systemach cyfrowych*, WBTC, Warszawa 2004

9. Bogusz J.: *Programowanie mikrokontrolerów 8051 w języku C w praktyce*, WBTC, Warszawa 2005

10. Daca W.: *Mikrokontrolery od układów 8-bitowych do 32-bitowych*, MIKOM, Warszawa 2000

11. Doliński J.: *Mikrokontrolery AVR w praktyce*, WBTC, Warszawa 2004

12. Doliński J.: *Mikrokomputer jednoukładowy INTEL 8051*, Wydawnictwo PLJ, Warszawa 1993

13. Filipkowski A.: *Układy elektroniczne analogowe i cyfrowe*, WNT, Warszawa 1978

14. Frei M.: *Samochodowe magistrale danych w praktyce warsztatowej*, WKŁ, Warszawa 2010

15. Fryśkowski B., Grzejszczuk E.: *Systemy transmisji danych*, WKŁ, Warszawa 2010

16. Gałka P., Gałka P.: *Podstawy programowania mikrokontrolera 8051*, MIKOM, Warszawa 1995

17. Gawrysiak M.: *Mechatronika i projektowanie mechatroniczne*, Wydawnictwo Politechniki Białostockiej, Białystok 1997

18. Głocki W.: *Układy cyfrowe*, WSiP, Warszawa 2005

19. Górecki P.: *Mikrokontrolery dla początkujących*, WBTC, Warszawa 2006

20. Hadam P.: *Projektowanie systemów mikroprocesorowych*, WBTC, Warszawa 2004

21. Jabłoński T.: *Mikrokontrolery PIC16F8x w praktyce*, WBTC, Warszawa 2002

22. Wilkinson B.: *Układy cyfrowe*, WKŁ, Warszawa 2000

23. Kardaś M.: *Mikrokontrolery AVR język C podstawy programowania*, Wydawnictwo ATNEL, Szczecin 2011

24. Kernighan B. W., Ritchie D. M.: *Język ANSI C*, WNT, Warszawa 2000

25. Kisiel R., Bajera A.: *Podstawy konstruowania urządzeń elektronicznych*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 1999

26. Kowalik R., Pawlicki C.: *Podstawy teletechniki dla elektryków*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2006

27. Lewis D. W.: *Programowanie: między asemblerem a językiem C. Podstawy oprogramowania wbudowanego*, Wydawnictwo RM, Warszawa 2004

28. Majewski J.: *Programowanie mikrokontrolerów 8051 w języku C, pierwsze kroki*, WBTC, Warszawa 2005

29. Majewski J., Kardach K.: *Programowanie mikrokontrolerów z serii 8x51 w języku C*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2002

30. Mielczarek W.: *Szeregowe interfejsy cyfrowe*, Wydawnictwo HELION, Gliwice 1993

31. Misiurewicz P.: *Podstawy techniki mikroprocesorowej*, WNT, Warszawa1991

32. Małysiak H., Pochopień B., Wróbel E.: *Mikrokomputery klasy IBM PC*, WNT, Warszawa 1992

33. Misiurewicz P.: *Układy mikroprocesorowe struktury i programowanie*, WNT, Warszawa1983

34. Owczarek T.: *Laboratorium podstaw techniki mikroprocesorowej i elementów konstrukcji systemów cyfrowych*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 1999

35. Pełka R.: *Mikrokontrolery: architektura, programowanie, zastosowania*, WKŁ, Warszawa1999

36. Praca zbiorowa: *Modułowe systemy mikrokomputerowe*, WNT, Warszawa 1984

37. Praca zbiorowa: *Wybrane układy MOS-LSI zastosowania pomiary*, WKŁ, Warszawa 1983

38. Rydzewski A.: *Mikrokomputery jednoukładowe rodziny MCS-51*, WNT, Warszawa 1992

39. Rydzewski A., Sacha K.: *Mikrokomputer elementy, budowa, działanie*, Wydawnictwo Czasopism i Książek Technicznych NOT-SIGMA, Warszawa 1985

40. Sacha K.: *Pamięci półprzewodnikowe RAM*, WNT, Warszawa 1991

41. Sacha K.: *Systemy czasu rzeczywistego*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2006

42. Simmonds A.: *Wprowadzenie do transmisji danych*, WKŁ, Warszawa 1999

43. Szymczyk P.: *Systemy operacyjne czasu rzeczywistego*, Kraków, Wydawnictwa AGH, 2003

44. Tondo C. L., Gimpel S. E.: *Język ANSI C programowanie, ćwiczenia*, Wydawnictwo HELION, Gliwice 2010

45. Wiązania M.: *Programowanie mikrokontrolerów AVR w języku BASCOM*, WBTC, Warszawa 2004

46. Witkowski A.: *Mikrokontrolery AVR programowanie w języku C przykłady zastosowań*, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 2006

47. Wojtuszkiewicz K.: *Urządzenia techniki komputerowej, część I, Jak działa komputer*, MIKOM, Warszawa 2002

48. Wojtuszkiewicz K.: *Urządzenia techniki komputerowej, część II, Urządzenia peryferyjne i interfejsy*, MIKOM, Warszawa 2000

49. www.btc.pl/pdf/zl2avr.pdf: *ZL2AVR – zestaw uruchomieniowy dla mikrokontrolerów AVR ATmega8*, WBTC, Warszawa 2004

50. www.btc.pl/pdf/zl2prg.pdf: *ZL2PRG – programator ISP dla mikrokontrolerów AVR firmy Atmel*, WBTC, Warszawa 2004