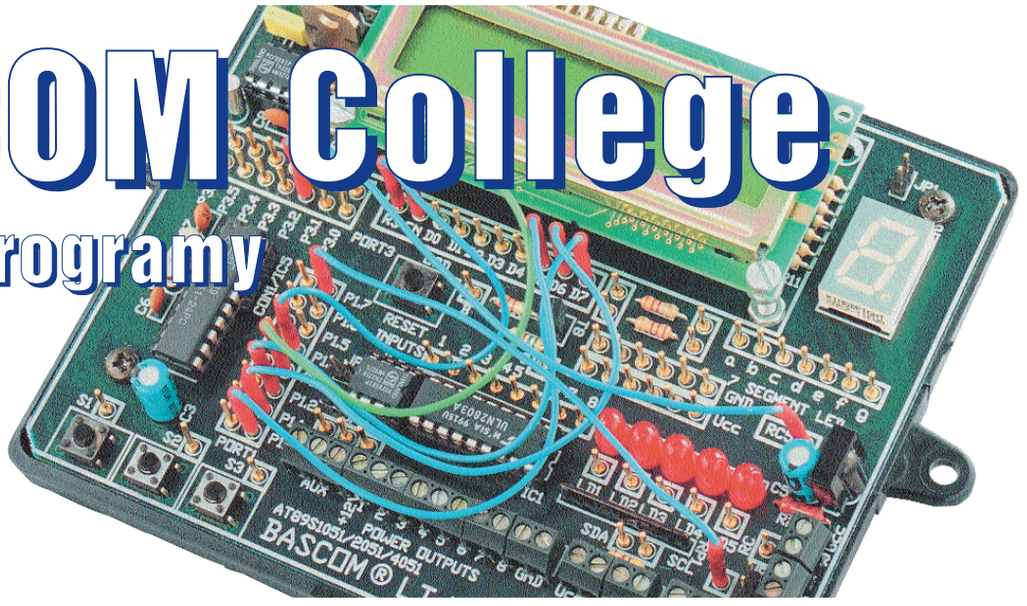
BASCOM College

Pierwsze programy



Ćwiczenie

Witam ponownie Studentów BASCOM College. W wykładzie wchłonęliśmy już sporą dawkę teorii dotyczącej wyświetlaczy LCD, wypróbowaliśmy wbudowany w pakiet BASCOM symulator programowy i najwyższa już pora na bliższe zapoznanie się z jednym z najważniejszych elementów naszego systemu edukacyjnego - płytką testową. Podczas tego ćwiczenia poznamy także praktycznie podstawowe polecenia odnoszące się do operacji dokonywanych na portach i pojedynczych wyprowadzeniach procesora 89C2051. Jeżeli zostanie nam dość czasu, to spróbujemy także emulacji sprzętowej wyświetlacza LCD.

Zacznijmy od bliższego zapoznania się z płytką testową AVT-2500. Zanim jednak przejdziemy do tej części ćwiczenia, chciałbym wyjaśnić pewne nieporozumienie, które powstało w związku z konstrukcją naszej płytki. Od kilku dociekliwych Czytelników otrzymałem e-maile informujące o błędach na płytce testowej! Błędy te miały polegać na nieumieszczeniu na schemacie złącza oznaczonego jako ISP oraz jumpera JP7. Rzeczywiście, elementy te zostały **celowo** pominięte na schemacie, ponieważ nie chciałem bez potrzeby komplikować rysunku przez wprowadzanie do niego dodatkowych, potrzebnych dopiero w dalekiej przyszłości składników.Zakładam, że przeszliście już przez męki związane z wlutowaniem w płytkę testową blisko 100 kołeczków, że nie poparzyliście się podczas podgrzewania izolacji termokurczliwej na przewodach montażowych i że gotowa płytka testowa leży przed Wami na biurku. Pozostaje więc już tylko dołączyć ją do komputera oraz zasilania i rozpocząć pracę.

Uwaga! Dołączanie płytki do komputera może odbywać się wyłącznie przy odłączonym zasilaniu obydwóch urządzeń. Niespełnienie tego warunku może doprowadzić do uszkodzenia portu równoległego, a tym samym płyty głównej nowoczesnego komputera!

Operacje na portach i pojedynczych pinach procesora 89C2051

W struktury procesorów podrodziny 89CX051 wbudowane są dwa ośmiobitowe porty wejściowo-wyjściowe nazwane, P1 i P3. Zapewnia to procesorowi możliwość komunikacji ze światem zewnętrznym, przyjmowanie z niego potrzebnych informacji, a także wysyłanie danych do układów peryferyjnych i sterowanie urządzeniami podporządkowanymi. Do dyspozycji mamy piętnaście wejść i wyjść, które możemy wykorzystywać zgodnie z aktualnymi potrzebami. Z pewnością wielu Czytelników zwróciło uwagę na drobną nieścisłość w poprzednim zdaniu. Najpierw mówiliśmy o dwóch ośmiobitowych portach, a następnie o piętnastu wejściach! Na szczęście nie była to pomyłka: procesor 89C2051 rzeczywiście posiada dwa pełne porty ośmiobitowe, z tym jednak, że wyjście 6 portu P3 nie jest dostępne z zewnątrz. Jest to wyjście komparatora analogowego wbudowanego w procesor, a jego stan, a tym samym informację o relacji pomiędzy napięciami dołączonymi do wejść komparatora, możemy odczytać na drodze programowej.

Sądzę, że warto uporządkować sobie informacje na temat portów procesora ‘2051, tym bardziej, że wiele ich pinów spełnia dodatkowe, niekiedy bardzo użyteczne funkcje.

PORT P1				
Pin portu	*Pin układu*	*Funkcja*	*Funkcja dodatkowa*	*Uwagi*
P1.0	12	IN/OUT	Wejście „+” komparatora analogowego	Brak rezystora podciągającego
P1.1	13	IN/OUT	Wejście „-” komparatora analogowego	Brak rezystora podciągającego
P1.2	14	IN/OUT	Brak	
P1.3	15	IN/OUT	Brak	
P1.4	16	IN/OUT	Brak	
P1.5	17	IN/OUT	Brak	
P1.6	18	IN/OUT	Brak	
P1.7	19	IN/OUT	Brak	

PORT P3				
Pin portu	*Pin układu*	*Funkcja*	*Funkcja dodatkowa*	*Uwagi*
P3.0	2	IN/OUT	Wejście RXD toru transmisji RS232	
P3.1	3	IN/OUT	Wejście TXD toru transmisji RS232	
P3.2	6	IN/OUT	Wejście przerwania zewnętrznego INT0	Aktywne opadającym zboczem lub poziomem
P3.3	7	IN/OUT	Wejście przerwania zewnętrznego INT1	Aktywne opadającym zboczem lub poziomem
P3.4	8	IN/OUT	Wejście timera 0	
P3.5	9	IN/OUT	Wejście timera 1	
P3.6	-	IN/OUT	Wewnętrzne wyjście komparatora analogowego	Odczyt stanu tego wyjścia jest możliwy tylko na drodze programowej
P3.7	11	IN/OUT	Brak	

Bardzo ważna uwaga!

Dwa wejścia portu P1: P1.0 i P1.1 nie zostały wyposażone w wewnętrzne rezystory podciągające! Jeżeli jedno z tych wyjść ustawimy w stan niski, to będzie ono zachowywać się dokładnie tak, jak pozostałe aktywne wyprowadzenia procesora: wewnętrzny rezystor wymusi na nim stan wysoki. Jeżeli jednak ustawimy P1.0 lub P1.1 w stan wysoki, to wyjście to bez stosowania elementów zewnętrznych będzie “wisieć w powietrzu” i nie będzie mogło być wykorzystane np. do sterowania bazy tranzystora lub innego wejścia bez rezystora podciągającego.

Na razie zajmiemy się tylko podstawowymi funkcjami pełnionymi przez porty procesora 89C2051. Funkcje dodatkowe będziemy poznawać stopniowo, w miarę zdobywania kolejnych porcji wiedzy o programowaniu w języku MCS BASIC.

Na **rysunku 1** został pokazany schemat połączeń, jakie będziemy musieli wykonać w celu przerobienia materiału dzisiejszych ćwiczeń, a **rysunek 2** pokazuje schemat montażowy: połączenia, które będziemy musieli wykonać na płytce testowej w celu wykonania pierwszych ćwiczeń.

Wszystkie ćwiczenia będziemy mogli przerobić na trzech "poziomach": za pomocą emulatora programowego, emulatora sprzętowego oraz wykorzystując zaprogramowany procesor.

Operacje na portach procesora '2051 możemy przeprowadzać wykorzystując polecenia programowe odnoszące się bądź do całego portu, bądź do pojedynczych pinów. Najprostszymi poleceniami, za pomocą których możemy zmienić stan pojedynczego wyprowadzenia są:

SET [port.pin] oraz RESET [port.pin]

Wydanie polecenia SET powoduje ustawienie wyznaczonego pinu w stan wysoki.
Wydanie polecenia RESET powoduje ustawienie wyznaczonego pinu w stan niski.
Np.:
SET P3.0 spowoduje wystąpienie stanu wysokiego na wyprowadzeniu 0 portu P3

RESET P3.1 spowoduje wystąpienie stanu niskiego na wyprowadzeniu 1 portu P3

Jeszcze prościej możemy dokonywać operacji odnoszących się do całego portu.

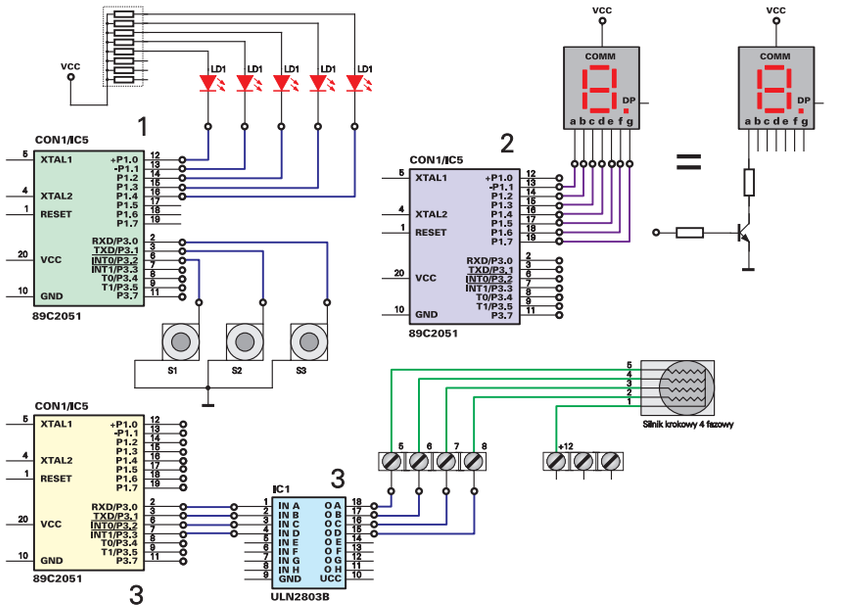
P[1,3] = [X (liczba z zakresu 0 ...255)]

Wydanie tego polecenia powoduje wysłanie binarnej reprezentacji liczby X na wyjścia wskazanego portu

Np.
P3 = 0 spowoduje ustawienie wszystkich wyjść portu P3 w stan niski
P3 = 255 spowoduje ustawienie wszystkich wyjść portu P3 w stan wysoki.
P3 = 3 spowoduje wysłanie na wyjścia portu P3 liczby "3", czyli ustawienie w stan wysoki wyjść P3.0 i P3.1.

Sprawdźmy teraz w praktyce, czy rzeczywiście podane polecenia funkcjonują poprawnie. Uruchamiamy program BASCOM i w okienku edytora piszemy sobie następujący programik:

```
P3= 0
End
```



Rys. 1

To chyba najkrótszy program, jaki kiedykolwiek został napisany, ale jego działanie nie jest pozbawione sensu. Kompilujemy nasz program (za pomocą naciśnięcia klawisza F7 możemy podać dowolną nazwę pliku lub pozostawić "NONAME"), a następnie naciskamy klawisz F2 wywołując w ten sposób panel symulatora sprzętowego i programowego. Klawiszem oznaczonym strzałką uruchamiamy nasz program i pilnie obserwujemy okienko emulacji programowej, w którym zobrazowane są stany wyjść wszystkich portów procesora (także portów P0 i P2, które nie istnieją wprawdzie w procesorze '2051, ale których emulacja jest potrzebna przy korzystaniu z "większych" '51).

Zgodnie z przewidywaniami, po wykonaniu naszego programu wszystkie wyjścia portu P3 znalazły się w stanie niskim, co sygnalizowane jest wyłączeniem wszystkich symbolizujących je lampek (rys. 3). Powtórzmy nasze doświadczenie kilkakrotnie, cały czas obserwując ekran emulatora programowego. Sądzę, że wszyscy z Was zauważyli już pewne dziwne zjawisko, występujące po każdorazowym uruchomieniu naszego programu. Otóż, lampki obrazujące stan wyjść portu P3 zapalają się na moment, aby następnie, zgodnie z poleceniem programowym zgasnąć. Oznacza to, że po uruchomieniu wszystkie wyjścia portu P3 przyjmują na moment stan wysoki, a dopiero po chwili, zgodnie z wydanym poleceniem, stan niski. Co jest, czyżby błąd w programie emulatora? Nic podobnego, emulator oddaje wiernie wszystkie czynności wykonywane przez procesor, a nie tylko te, których wykonanie zostało przewidziane w testowanym programie. Aby wykonywanie programu mogło się rozpocząć, procesor przez okres co najmniej dwóch taktów zegarowych musi znajdować się w stanie RESET, co połączone jest z wystąpieniem na wszystkich wyjściach portów stanu wysokiego. Dopiero po ustąpieniu stanu wysokiego z wejścia RESET procesora, może on rozpocząć normalną pracę.

Uruchommy teraz jeszcze raz nasz programik obserwując tym razem płytkę testową. Aha, zapomniałem powiedzieć, że symulator sprzętowy i programowy mogą działać jednocześnie. Spowalnia to trochę pracę programu i nie zawsze ma sens, ale w przypadku prostych ćwiczeń nie ma to większego znaczenia. Od razu zauważymy, że zaszły na niej zjawiska z pozoru odwrotne do tych, jaki obserwowaliśmy w okienku emulatora programowego. Wszystkie diody LED najpierw zgasły na moment, a potem zapaliły się ponownie. Jednak i tym razem wszystko jest w porządku:

Diody LED na naszej płytce testowej włączone są pomiędzy wyjścia procesora (lub emulatora sprzętowego) a plus zasilania i świecą przy stanie niskim dołączonego do nich wyjścia.

Poćwiczmy jeszcze trochę wysyłanie danych do portów procesora i ożywmy trochę naszą płytkę testową. Piszemy kolejny, króciutki programik:

```
$sim        'zawiadomienie kompilatora, że program będzie testowany w symulacji 'sprzętowej lub programowej
Dim R As Byte                   'deklaracja zmiennej R
For R = 1 To 255    'od wartości R = 1 do osiągnięcia wartości R=255 powtarzaj:
P3 = R                          'wyślij do portu P3 aktualną wartość R
Next R                          'powtórz powyższą instrukcję
End                             'po wyjściu z pętli FOR .... NEXT koniec programu
```

i po skompilowaniu uruchamiamy go w symulacji programowej i sprzętowej. Efekt jest nawet ładny: wszystkie diody zapalają się i gasną, wyczerpując wszystkie możliwe kombinacje kodu dwójkowego z zakresu od 1 do 255 w okienku emulatora sprzętowego i od 1 do 32 na płytce testowej, na której nie zmieściła się już większa liczba diod.

Jak jednak będziemy musieli postąpić, jeżeli naszym zamiarem będzie zmiana stanu tylko jednego wyprowadzenia któregoś z portów, bez ingerowania w stan pozostałych wejść lub wyjść? To bardzo proste, napiszmy sobie mały program:

```
P3 = 0                    'wyślij na wyjścia portu P3 liczbę "0"
Do
Set P3.1   'ustaw stan wysoki na zadanym wyjściu portu (P3.1)
Reset P3.1'ustawia stan niski na zadanym wyjściu portu (P3.1)
Loop
```

I jak zwykle: F7, F2 i już możemy oglądać rezultaty działania poleceń SET i RESET. Zgodnie z naszymi oczekiwaniami dioda dołączona do wyjścia 1 portu P3 migocze, a pozostałe pozostają wyłączone. Możliwość dokonywania operacji na pojedynczych wyjściach portów ma ogromne znaczenia praktyczne i upraszcza pisanie programów, o czym przekonacie się w najbliższej przyszłości.

Jak dotąd, wszystkie wykonane ćwiczenia nie miały większego zastosowania użytkowego i służyły wyłącznie celom poznawczym. Spróbujmy teraz zrobić coś, co nie tylko będzie ćwiczeniem szkoleniowym, ale znajdzie praktyczne zastosowanie: uruchommy wyświetlacz siedmiosegmentowy na płytce testowej i w emulacji programowej.

Popatrzmy przez chwilę na schemat naszej płytki testowej zamieszczony w numerze 3/2000 Elektroniki dla Wszystkich i na samą płytkę. W układzie został zastosowany wyświetlacz siedmiosegmentowy ze wspólną anodą, a do wejść każdego z segmentów został dołączony tranzystor NPN, który po spolaryzowaniu bazy będzie zwierał odpowiadający mu segment do masy, włączając go. A więc, podając na odpowiednie wejścia, oznaczone na płytce jako a ... g, stan wysoki możemy uzyskać włączanie odpowiednich segmentów wyświetlacza i obrazować na nim cyfry, a także inne znaki możliwe do zdefiniowania za pomocą siedmiu świecących segmentów. Jeżeli zatem dołączymy wejścia a .... g do wyjść jednego z portów procesora, to wystarczy wysyłać do tego portu odpowiednie liczby, odpowiadające kodowi wyświetlacza siedmiosegmentowego, aby uzyskać wyświetlanie żądanych cyfr. A więc, do dzieła: zmontujmy na płytce testowej układ przedstawiony na rysunku 1 - 2 i na schemacie montażowym z **rysunku 4** i zastanówmy się, jakie liczby będziemy musieli wysyłać do portu P3 procesora, aby uzyskać wyświetlanie cyfr od 0 do 9.

Przeanalizujmy najprostszy przypadek: cyfrę "1". Do jej wyświetlenia niezbędne jest włączenie segmentów "b" i "c" wyświetlacza, a pozostałe segmenty muszą być wyłączone. Segment "b" sterowany jest z wyjścia P3.1, a segment "c" z wyjścia P3.2 procesora. A zatem liczba, której podanie na wyjścia portu P3 spowoduje ukazanie się na wyświetlaczu upragnionej jedynki to binarnie:

0X00 0110

gdzie X - bit nieznaczący (pamiętajmy, że wyjście P3.6 nie może być w jakikolwiek sposób sterowane od "strony" procesora i nie jest wyprowadzone na zewnątrz kostki. Liczba binarna 0000 0110 to w kodzie dziesiętnym po prostu "6". A zatem piszemy:

```
p3= 6
End
```

i kompilujemy ten wyjątkowo „rozbudowany" program. Następnie możemy uruchomić go w symulacji sprzętowej i/lub programowej, ale najpierw musimy wykonać jeszcze jedną czynność. Z pewnością zauważyliście rysunek wyświetlacza siedmiosegmentowego w okienku symulatora programowego. Nie jest to bynajmniej ozdoba, ale kolejny "fajerwerk" pakietu BASCOM, ułatwiający programistom życie. Jednak aby z niego skorzystać, musimy go najpierw odpowiednio skonfigurować, czyli przypisać odpowiednie segmenty emulowanego wyświetlacza odpowiadającym im wyprowadzeniom portu procesora. Naprowadzamy zatem kursor na rysunek wyświetlacza i klikamy PRAWYM klawiszkiem myszki. Następstwem tego odważnego kroku jest rozwinięcie się okienka konfiguracyjnego wyświetlacza, pokazanego na **rysunku 5**. Następnie wpisujemy w tabelkę zawartą w tym okienku odpowiednie wartości, zgodnie z tabelą:

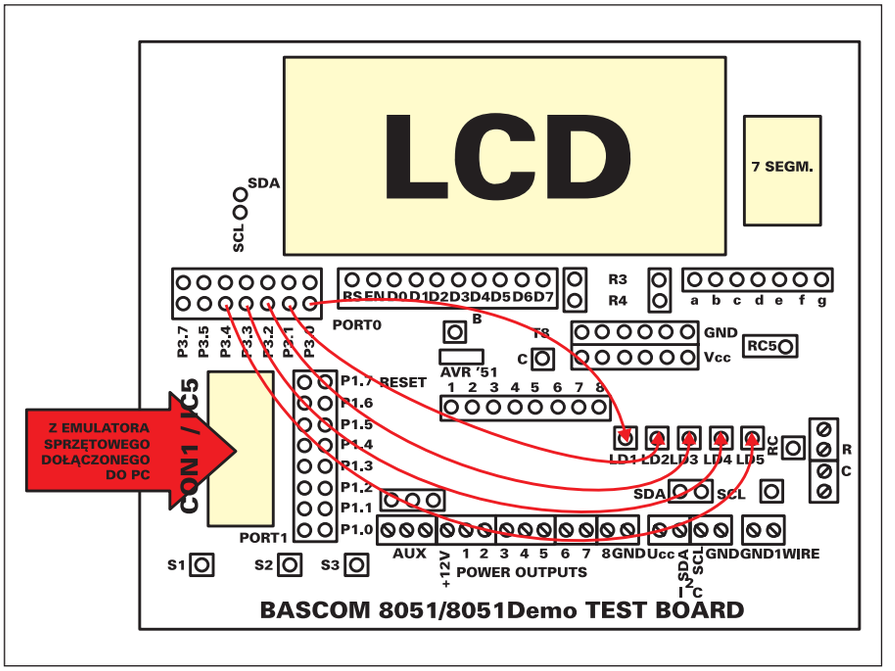
Segment	*Pin*
A	P3.0
B	P3.1
C	P3.2
D	P3.3
E	P3.4
F	P3.5
G	P3.7

Klikamy "OK" w okienku konfiguracyjnym i uruchamiamy program. BINGO! Na wyświetlaczu ukazała się cyfra "1" i wszystko wskazuje na to, że dokładnie tak samo zachowa się wyświetlacz na naszej płytce testowej, oczywiście po włączeniu symulacji sprzętowej i ponownym uruchomieniu programu!

Zajmijmy się teraz określeniem, jakie wartości należy wysłać do portu P3, aby uzyskać wyświetlanie pozostałych dziewięciu cyfr. Wykonałem tę pracę za Was, a jej wynik pokazany został w poniższej tabeli. Pozostaje nam zatem praktyczne sprawdzenie poprawności moich obliczeń (wykonanych dla przyśpieszenia pracy w arkuszu kalkulacyjnym MS EXCELL). Bierzmy się zatem do napisania pierwszego nieco bardziej rozbudowanego programu.

Segment	*G*	*F*	*E*	*D*	*C*	*B*	*A*	*Wartość*
Cyfra								
0	0	1	1	1	1	1	1	63
1	0	0	0	0	1	1	0	6
2	1	0	1	1	0	1	1	155
3	1	0	0	1	1	1	1	143
4	1	1	0	0	1	1	0	166
5	1	1	0	1	1	0	1	173
6	1	1	1	1	1	0	1	189
7	0	0	0	0	1	1	1	7
8	1	1	1	1	1	1	1	191
9	1	1	0	1	1	1	1	175

Rys. 2



Pierwszym problemem, jaki będziemy musieli rozwiązać, będzie z pewnością sprawa uporządkowania obliczonych wartości tak, abyśmy mogli z nich korzystać w możliwie wygodny sposób. Zastanówmy się, w jakich programach będziemy musieli korzystać z wyświetlania cyfr? Ano, przede wszystkim we wszelkiego rodzaju licznikach, zegarach, miernikach częstotliwości i innych podobnych urządzeniach. W większości wypadków będziemy zatem mieli do czynienia z cyklicznym wyświetlaniem cyfr. Oczywiście, możemy każdej z wartości nadać indywidualne nazwy, np. Cyfra1, Cyfra2 itd. Wątpię jednak, czy korzystanie z tak określonych stałych byłoby wygodne! Postąpimy zatem inaczej: umieścimy nasze wartości w tabeli, czyli mówiąc językiem programistów zadeklarujemy tablicę danych. Sposób deklaracji tablicy jest bardzo podobny do deklarowania pojedynczej zmiennej, a różnica polega na tym, że tym razem powiadamiamy kompilator o konieczności zarezerwowania w pamięci RAM procesora miejsca jednorazowo na więcej niż jedną wartość. Mówiąc ściślej, deklaracja tablicy polega na zadeklarowaniu kilku zmiennych o tej samej nazwie, ale o innym "numerze porządkowym". Jak wygodnym posunięciem jest zadeklarowanie tablicy danych, przekonamy się za chwilę, ale pamiętajcie o jednej sprawie:

Każda zmienna typu "BYTE" zajmuje w pamięci procesora dokładnie jeden bajt, których do dyspozycji mamy zaledwie 128. Inne zmienne zajmują (z wyjątkiem zmiennej typu "BIT") jeszcze więcej miejsca. Deklarowanie tablic powoduje drastyczne zwiększenie obszaru zajmowanej pamięci i dlatego musimy stosować je z rozwagą, deklarując tablice o minimalnych dopuszczalnych rozmiarach.

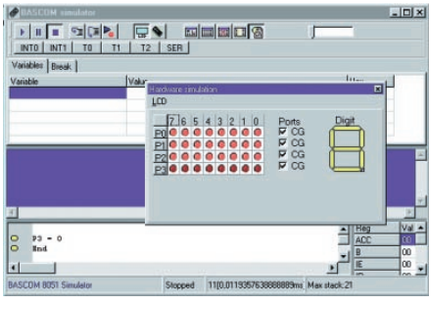
Pomimo powyższego ostrzeżenia, dochodzimy jednak do wniosku, że zadeklarowania tablicy jest niezbędne i czynimy to za pomocą polecenia:

DIM [zmienna] ([ilość stosowanych zmiennych]) As [typ zmiennej]

W naszym, konkretnym przypadku polecenie to będzie miało postać:

DIM Cyfra(9) As Byte

Rys. 3



Po wydaniu tego polecenia kompilator zarezerwuje sobie w pamięci już nie pojedynczą komórkę, ale jakby malutką szafkę z dziesięcioma (pierwsza szafka ma numer "0") szufladkami, w których umieścimy liczby potrzebne do wyświetlenia wszystkich dziesięciu cyfr.

No dobrze, mamy już tablicę danych, ale niewielki z niej pożytek, tak jak z każdej pustej szafki. A zatem zapełnijmy naszą tablicę potrzebnymi nam danymi. Czynność tę możemy wykonać w najrozmaitszy sposób, ale na razie wybierzemy metodę najprostszą, wręcz intuicyjną. Piszemy:

i po uruchomieniu programu nasza szafka zostanie zapełniona aż po same brzegi! Pamiętajmy jednak, że zużyliśmy już 10 bajtów cennej pamięci RAM procesora. Nie obawiajcie się jednak, to co zostało wystarczy jeszcze na potrzeby nawet dość rozbudowanego programu, a w najbliższej przyszłości dowiemy się, jak można usunąć niepotrzebne już zmienne z pamięci RAM i uwolnić zajmowaną przez nie pamięć.

```
Dim Cyfra(9) As byte

Cyfra(0) = 63
Cyfra(1) = 6
Cyfra(2) = 155
Cyfra(3) = 143
Cyfra(4) = 166
Cyfra(5) = 173
Cyfra(6) = 189
Cyfra(7) = 7
Cyfra(8) = 191
Cyfra(9) = 175
```

Podprogram, który napisaliśmy przed chwilą warto zapisać sobie w osobnym pliku. Z pewnością przyda się on nam w przyszłości. Tu na marginesie drobna uwaga: dobrą praktyką jest zapisywanie w osobnym katalogu szczególnie cennych fragmentów programów i ciekawych procedur. Po jakimś czasie uzbieramy sobie pokaźną bibliotekę, a pisanie kolejnego programu często będzie sprowadzać się do "sklejania" ze sobą zarchiwizowanych uprzednio podprogramów.

Jednak jak na razie nie mamy zbyt wielkiego pożytku z napisanego programu. Po skompilowaniu i uruchomieniu załaduje on wprawdzie potrzebne dane do pamięci i nic więcej. A więc, musimy jeszcze chwilę popracować, aby wreszcie ujrzeć wszystkie cyfry ukazujące się na wyświetlaczu. Zakładamy, ze wystarczy nam jednorazowe sprawdzenie poprawności przeprowadzonych uprzednio operacji. Piszemy zatem kolejne linijki programu:

```
For R = 0 To 9
P3 = Cyfra(r)
Next R
P3= 0
End
```

A na początku, zaraz po poleceniu Dim Cyfra(9) As Byte dopisujemy:

Dim R as Byte

Działanie polecenia Dim jest już dla Was oczywiste, ale zastanówmy się, jakie czynności mają wykonywać ostanie cztery linijki programu.

For R = 0 To 9 oznacza, że wszystkie czynności zawarte pomiędzy tym poleceniem a NEXT mają być wykonane dziesięć razy, a po każdym ich wykonaniu wartość R zostanie powiększona o 1.

Polecenie P3 = Cyfra(r) wykonane zostanie dziesięć razy, a za każdym razem do portu P3 zostanie wysłana liczba przyporządkowana aktualnej wartości zmiennej R, czyli kolejno wszystkie umieszczone w tablicy liczby.

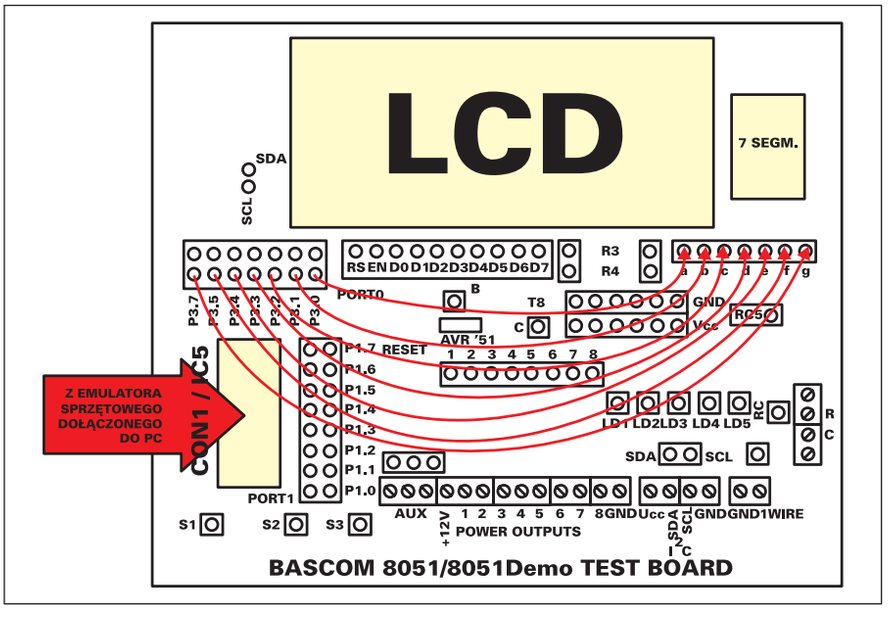
Polecenie NEXT zamyka pętlę programową

Polecenie P3=0 sprawia, że po wyświetleniu wszystkich cyfr wyświetlacz siedmiosegmentowy zostanie wyłączony (na wyjściach portu P3 pojawią się same zera).

End - koniec programu.

Wygląda na to, że wszystko powinno działać zgodnie z przewidywaniami. Kompilujemy więc nasz program i włączamy symulację

Rys. 4



programową. Tym razem jednak nie bardzo mamy ochotę zakrzyknąć "BINGO!". Na symulowanym wyświetlaczu coś się wprawdzie pokazało, ale wyświetlanie cyfr przebiegło tak szybko, że nawet nie zdążyliśmy zaobserwować jego ewentualnej poprawności. W symulacji sprzętowej, na płytce testowej sytuacja wygląda równie źle: wyświetlacz zamigotał i wszystkie segmenty wyłączyły się. Gdzie więc tkwi błąd?

Błąd tkwi w zbyt dużej szybkości pracy programu, nawet podczas powolnej emulacji sprzętowej czy programowej. Gdybyśmy teraz tak napisany program załadowali do procesora i uruchomili, to zobaczylibyśmy co najwyżej krótki błysk segmentów wyświetlacza. Na szczęście rozwiązania problemu są bardzo proste: wystarczy w jakiś sposób zmusić program, aby po wyświetleniu każdej cyfry zaczekał trochę na powolnego człowieka i umożliwił mu odczytanie pokazanej cyfry. Celowo napisałem "rozwiązania", ponieważ żądane opóźnienie zrealizujemy dwoma różnymi metodami, odpowiednimi dla symulacji i pracy programu w zaprogramowanym procesorze.

Realizowanie opóźnienia w symulacji sprzętowej i/lub programowej

W symulacji najprościej jest zrealizować opóźnienie za pomocą dodatkowej pętli programowej wstawionej w odpowiednim miejscu programu. Pętla może być pusta, nie musi wykonywać jakichkolwiek czynności, wystarczy, że wykonanie jej samej zajmuje trochę czasu. Najprościej zrealizować taką pętlę za pomocą znanej już Wam instrukcji NEXT ... FOR. Ilość powtórzeń pętli zależy od szybkości działania procesora w komputerze i najlepiej ustalić ją doświadczalnie. Przykładowo: wykonanie instrukcji:

For X = 1 to 100
Next X

zajmuje na komputerze z procesorem PENTIUM 133 około 2 sekund.

Dopisujemy więc do naszego programu pętlę opóźniającą, nie zapominając o zadeklarowaniu kolejnej zmiennej: X. Zarówno deklaracja tej zmiennej, jak i pętla programowa zostaną przed zaprogramowaniem procesora usunięte z program, tak więc nie musimy martwić się zwiększeniem zajmowanej powierzchni pamięci RAM.

Ostateczna, gotowa do uruchomienia wersja naszego programu wygląda następująco:

```
                                    'Deklaracje Zmiennych
Dim Cyfra(9) As Byte          'deklaracja tablicy zmiennych
Dim R As Byte                 'deklaracja zmiennej pomocniczej  (licznik instrukcji FOR ...NEXT)
Dim X As Byte                  'deklaracja zmiennej pomocniczej opóźnienia
'Ładowanie danych do tablicy
Cyfra(0) = 63                 'umieszczenie w tablicy kodu cyfry "0"
Cyfra(1) = 6                  'umieszczenie w tablicy kodu cyfry "1"
Cyfra(2) = 155                'umieszczenie w tablicy kodu cyfry "2"
Cyfra(3) = 143                'umieszczenie w tablicy kodu cyfry "3"
Cyfra(4) = 166                'umieszczenie w tablicy kodu cyfry "4"
Cyfra(5) = 173                'umieszczenie w tablicy kodu cyfry "5"
Cyfra(6) = 189                'umieszczenie w tablicy kodu cyfry "6"
Cyfra(7) = 7                  'umieszczenie w tablicy kodu cyfry "7"
Cyfra(8) = 191                'umieszczenie w tablicy kodu cyfry "8"
Cyfra(9) = 175                'umieszczenie w tablicy kodu cyfry "9"
'Właściwy program
Do                                  'wykonaj wszystko, co poniżej
For R = 0 To 9                'początek pętli głównej programu
  P3 = Cyfra(r)               'wysłanie do portu P3 wartości odpowiadającej cyfrze
    For X = 1 To 100          'pętla opóźnienia
    Next X                              'pętla opóźnienia
Next R                                  'zamknięcie pętli głównej
Loop                                    'wykonaj jeszcze raz
End
```

Zauważyliście pewnie, że dodałem do programu jeszcze jeden element: niekończącą się pętlę DO .... LOOP. W związku z tym nasz program będzie wykonywany w nieskończoność, wyświetlając na wyświetlaczu siedmiosegmentowym kolejne cyfry.

Realizacja opóźnienia w programie przeznaczonym do umieszczenia w procesorze

Język MCS BASIC oferuje nam wiele możliwości realizacji opóźnień czasowych, zwalniając nas **na razie** od konieczności poznawania obsługi timerów systemowych. Do dyspozycji mamy dwie podstawowe instrukcje, pozwalające na wnoszenie opóźnień z zakresu od 1 ms do 255 sekund, czyli do ponad 4 minut. Są to instrukcje:

WAIT [liczba sekund, liczba z zakresu 1 do 255)

WAITMS [liczba milisekund, liczba z zakresu od 1 do 255]

Po wydaniu jednaj z tych dwóch instrukcji program wstrzymuje swoje działanie aż do momentu upłynięcia oznaczonego czasu. Oczywiście, zawsze możliwe jest wydanie kolejno kilku instrukcji opóźnienia, chociażby w celu "załatania dziury" pomiędzy 255msek za 1 s. Np.:

Wait 1: Waitms 125
lub
Waitms 255: Waitms 100

Gdybyśmy chcieli teraz zaprogramować procesor i sprawdzić w "real world" jego działanie, to musimy pamiętać o bezwzględnym usunięciu z niego opóźnień realizowanych w formie pętli programowej i zastąpienie ich jednym z wyżej opisanych poleceń. Tak więc zamiast:

```
For X = 1 To 100
Next X
```

Wstawimy np.:

```
WAIT 1
```

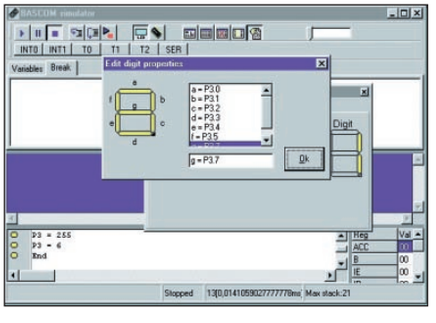
co da nam opóźnienie 1 sekundy.

Nie wyczerpaliśmy bynajmniej tematu wysyłania danych do portów procesora '51, ale pozostało już niewiele czasu z tej godziny lekcyjnej. W opisywaniu głównego tematu ćwiczenia przeszkadzały nam liczne dygresje, ale także dzięki nim wypełnialiśmy postawione przed nami zadania: uczyliśmy się języka MCS BASIC i obsługi pakietu BASCOM. Chciałbym wspomnieć Wam jeszcze, chociażby w największym skrócie, o odczytywaniu danych z portów procesora.

Odczytanie jakichkolwiek danych z portu procesora '2051 lub z pojedynczego pinu takiego portu jest możliwe dopiero po programowym ustawieniu na całym porcie lub wybranym pionie STANU WYSOKIEGO.

Wyprowadzenia portów procesora '2051 możemy z pewnym przybliżeniem traktować jako wyjścia typu OPEN DRAIN wyposażone w bufory, w których zatrzaskuje się podana z wewnątrz informacja. Jeżeli np. po podaniu polecenia:

Rys. 5



P3=0

chcielibyśmy odczytać z wejść tego portu jakiekolwiek dane, to niezależnie od stanów logicznych na dołączonych do nich układów peryferyjnych zawsze odczytywalibyśmy same zera! Dopiero po wydaniu polecenia:

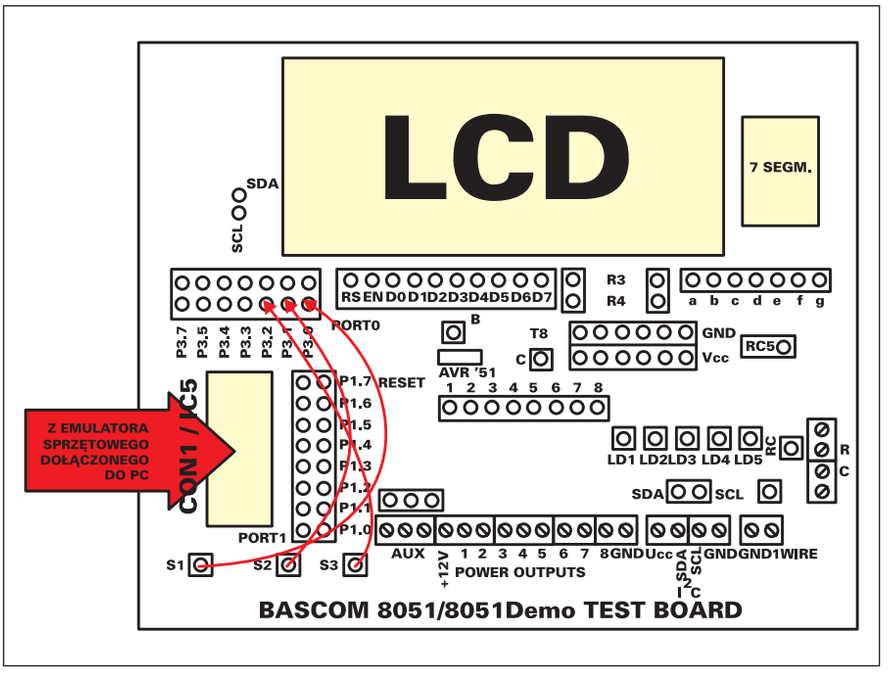
P3= 255

uzyskujemy dostęp do wejść portu P3.

W języku MCS BASIC liczby charakteryzujące stan portów procesora, a także ich pojedynczych pinów, możemy traktować jako zwykłe zmienne, tzn. nadawać im określone wartości, a także dokonywać wszelkich innych operacji dozwolonych na zmiennych.

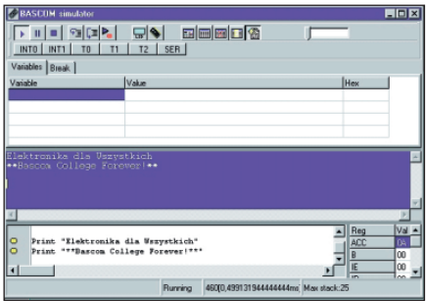
Np. po wydaniu polecenia A = P1 zmienna A przyjmie taką wartość, jaka znajduje się w momencie wydania polecenia na wejściach portu P1. Możemy także napisać: A = P1.3, a wtedy zmienna A przyjmie wartość, jaka aktualnie istnieje na wejściu 3 portu P1.

Napiszmy sobie zatem króciutki program:, a następnie zmontujmy na naszej płytce testowej układ pokazany na **rysunku 6**.

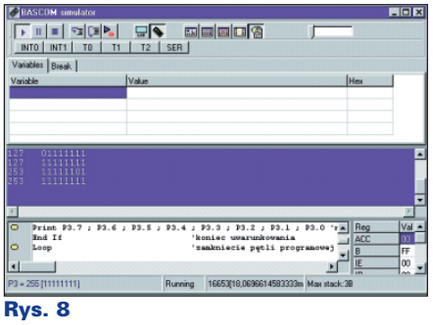


Rys. 6

```
Do                                          'początek niekończącej się pętli programowej
P3 = 255                                    'przygotuj wejścia portu P3 do odczytu danych
If P3 <> 255 Then                           'jeżeli wartość podana do P3 zmieniła się, to:
Print P3 ; "  " ;                           'napisz, jaka to jest wartość
Print P3.7 ; P3.6 ; P3.5 ; P3.4 ; P3.3 ; P3.2 ; P3.1 ; P3.0   'napisz, jakie są stany poszczególnych pinów
End If                                      'koniec uwarunkowania
Loop                                        'zamknięcie pętli programowej
```



Rys. 7



Rys. 8

Zanim jednak przejdziemy do testowania w symulacji sprzętowej naszego programu i zawartych w nim poleceń, winien jestem Wam pewne wyjaśnienia, dotyczące nowego dla nas polecenia PRINT, które znalazło się w powyższym programie.

Generalnie, polecenie **"PRINT**" używane jest do zapewnienia komunikacji pomiędzy komputerem PC a systemem mikroprocesorowym wyposażonym w interfejs RS232. W przypadku procesorów 'X051 i pracy w środowisku BASCOM-a nawiązanie takiej łączności jest szczególnie łatwe i będzie tematem jednej z następnych lekcji. Polecenie **"PRINT**" ma jednak jeszcze jedno, wyjątkowo użyteczne zastosowanie: podczas pracy w emulacji sprzętowej lub programowej umożliwia "podgląd" pracującego programu i w wielu przypadkach radykalnie przyspiesza i ułatwia jego uruchomienie.

W przypadku pracy w symulacji i bez wykorzystywania transmisji poprzez interfejs RS232 efekty działania polecenie PRINT kierowane są na ekran komputera, Napiszmy sobie dwie linijki programu i po skompilowaniu uruchommy go w symulatorze (**rys.7**):

```
Print "**Elektronika dla Wszystkich**"
Print "**BASCOM College Forever!**"
```

Tekst, którego wysłanie do portu szeregowego zostało zlecone poleceniem PRINT ukazał się na małym, niebieskim ekranie emulatora programowego i to samo zjawisko zajdzie w przypadku korzystania z emulatora sprzętowego. Na jednej z najbliższych lekcji pokażę Wam, jak bardzo polecenie PRINT może okazać się użyteczne, np. podczas uruchamiania programów obsługujących magistralę I^2C. Bez najmniejszej przesady: stosowanie tego polecenia do "podglądania" programu pracującego w symulacji może niejednokrotnie pozwolić na zaoszczędzenie wielu godzin czasu!

Wracajmy jednak do naszego programu testującego stan wejść portu P.3 procesora. Uruchamiamy go w symulacji sprzętowej, na zmontowanej wg rysunku 6 płytce testowej1. Początkowo nic się nie dzieje, ale po naciśnięciu któregokolwiek z przycisków S1 ... S3 na płytce na ekranie symulatora zaczynają pojawiać się liczby. Na początku zostaje wyświetlona liczba reprezentująca stan portu P3 w notacji dziesiętnej, a zaraz po niej ta sama liczba w kodzie binarnym.

Nie przerobiliśmy jeszcze całego zapowiedzianego materiału, a już odezwał się dzwonek oznajmiający koniec lekcji! Na zakończenie podam Wam trochę materiału do przerobienia w domu:

1. Spróbujcie napisać, skompilować i przetestować na płytce testowej poniższy program.

Ciąg dalszy na stronie 36.

Ciąg dalszy ze strony 28.

Będzie to jednocześnie praktycznie pierwsza próba skorzystania z wyświetlacza alfanumerycznego LCD umieszczonego na naszej płytce testowej. Schemat montażowy jest taki sam, jak do poprzedniego ćwiczenia.

2. Popatrzcie na główny schemat, na którym pozostał jeszcze nie omówiony fragment 3. Jeżeli posiadacie jakiś czterofazowy silnik krokowy (np. od starej stacji dysków 5,25”), to dołączcie go do naszej płytki w sposób pokazany na schemacie i spróbujcie napisać program wprawiający ten silnik w ruch. Nie mogłem sobie z tym problemem poradzić (no, może jest w tym trochę kokieterii) i bardzo byłbym wdzięczny za każdy pomysł na napisanie takiego programu przesłany mi na zbigniew.raabe@edw.com.pl.

```
$sim
Config Lcd = 16 * 1a
Cursor Off
Cls
Do
Set P3.0
Set P3.1
Set P3.2
If P3.0 = 0 Then
Cls
Print "Stan niski na P3.0"
Lcd "Low on P3.0"
End If
If P3.1 = 0 Then
Cls
Print "Stan niski na P3.1"
Lcd "Low on P3.1"
End If
If P3.2 = 0 Then
Cls
Print "Stan niski na P3.2"
Lcd "Low on P3.2"
End If
Loop
```

3. Jeżeli nie posiadacie silnika krokowego, to spróbujcie napisać i uruchomić program obsługujący zwykły silnik DC małej mocy. A może poeksperymentujecie z żarówkami na niskie napięcie lub innymi odbiornikami DC, dołączanymi do płytki testowej? Pamiętajcie jednak o jednym: do każdego z wyprowadzeń portów procesora ‘2051 może wpływać prąd nie większy niż 20mA i najlepiej zawsze stosujcie wzmacniacz prądowy z układem ULN2803 umieszczony na płytce testowej.

To wszystko, co chciałem przekazać Wam w ćwiczeniu 2.

Zbigniew Raabe
zbigniew.raabe@edw.com.pl