Mikrokontrolery? To takie proste...

Część 12
UART – port szeregowy procesora

Po dłuższej przerwie w opisie bloków funkcjonalnych mikrokontrolera 8051, wypełnioną opisem niezbędnych do programowania, instrukcji asemblera, kontynuujemy prezentację pozostałych układów wewnętrznych procesora. Dzięki temu, że już znacie język maszynowy procesora, a przynajmniej orientujecie się w jego składni i sposobach „panowania" na różnego rodzaju rejestrami kostki, będzie mi łatwiej opisywać elementy procesora, bowiem za każdym razem będę ilustrował opisy, przykładami sposobów programowania takich elementów jak port szeregowy, układy licznikowo-czasowe, czy układ przerwań. Cel takiego podejścia do tematu jest oczywisty, to praktyczne nauczenie Was, drodzy Czytelnicy, łatwego i przyjemnego korzystania z wszystkich możliwości mikrokontrolera 8051.

Zanim przejdę do sedna dzisiejszego odcinka, chciałbym zaanonsować wszystkim zainteresowanym Czytelnikom, że od dzisiejszego numeru EdW uruchamiam

„Kącik pocztowy 8051", w którym będę odpowiadał na wszystkie zawarte w waszych listach problemy, dotyczące programowania naszego mikrokontrolera. Ze względu na to, że część z ogromnej liczby listów które dostaję będzie wiązała się z wybranymi, prezentowanymi w kolejnym odcinku, problemami, odpowiedzi będą wiązały się z tematami, które już omawialiśmy lub tymi, które akurat są tematem kolejnego odcinka klasy mikroprocesorowej.

W dzisiejszej części zapoznamy się z układem transmisji szeregowej oraz praktycznymi sposobami programowania go i wykorzystywania do własnych celów, w tym także do przesyłania danych pomiędzy komputerem PC lub dwoma komputerkami edukacyjnymi.

Port szeregowy

Mikrokontroler 8051 i pochodne posiadają sprzętowy port szeregowy (w skrócie UART), dzięki któremu możliwe jest wysyłanie i odbieranie informacji w postaci szeregowej, czyli „bit po bicie". Jak już wiecie z opisu wyprowadzeń, który przedstawiłem na początku naszego kursu, procesor posiada dwie dedykowane końcówki które wchodzą w skład portu P3 procesora. Są to:

RXD – (P3.0) wejście szeregowe („**R**eceive data")
TXD – (P3.1) wyjście szeregowe („**T**ransmit data")

Jak zapewne pamiętacie, końcówki te mogą być wykorzystywane jako uniwersalne wejścia-wyjścia, dzięki instrukcjom zapisu do portu P3, np.

```
MOV     P3, #dana
```

lub indywidualnym sterowaniem każdej końcówki portu np.:

```
SETB    P3.0    ;ustawienie „1" na końcówce RXD
CLR     P3.1    ;ustawienie „0" na końcówce TXD
```

Jednak przy wykorzystaniu portu szeregowego, sterowanie końcówkami odbywa się automatycznie (za pomocą CPU), według ustawionych wcześniej przez programistę parametrów przesyłowych. Port szeregowy wysyła i odbiera dane w postaci bajtów (8-bitowych słów danych). Konwersja danej wysłanej lub odebranej przez procesor z postaci bajtu do postaci szeregowej lub odwrotnie, odbywa się automatycznie. Dzięki temu wystarczy wskazać tylko dana którą chcemy wysłać lub czekać na odbiór jej z zewnętrznego urządzenia, także wyposażonego w port szeregowy.

Miejscem z którego wysyła się wspomniane dane – bajty, lub do którego one trafiają po transmisji z zewnątrz jest specjalny rejestr, znajdujący się pod adresem 99h w pamięci wewnętrznej danych procesora w obszarze rejestrów specjalnych SFR. Rejestr ma nazwę SBUF a zapisać go można tak samo jak każdy inny rejestr, np. instrukcją zapisu poprzez wskaźnik Ri:

```
MOV     SBUF, @R1
```

W przypadku, kiedy wcześniej ustawiliśmy parametry transmisji i uruchomiliśmy port szeregowy (o tym jak to zrobić – za chwilę), taki zapis spowoduje automatyczne wytransmitowanie bajtu który wcześniej znajdował się pod adresem wskazywanym przez rejestr indeksowy R1.

W przypadku odbioru danej, po zakończeniu transmisji odebrany bajt informacji będzie automatycznie umieszczony w rejestrze SBUF, a fakt zajścia takiego zdarzenia zostanie zasygnalizowany w programie automatycznie. Dzięki temu będziemy wiedzieć, że w rejestrze SBUF czeka na odczytanie gotowa odebrana dana, z którą można zrobić na co się ma ochotę.

Zanim przejdę do omawiania sposobu sterowania transmisją danych poprzez port szeregowy, pragnę wyjaśnić dwa pojęcia związane z portem szeregowym. Być może niektórzy z Was dokładnie wiedzą o co chodzi, lecz pozostałym Czytelnikom należy się wyjaśnienie zasady samej transmisji szeregowej.

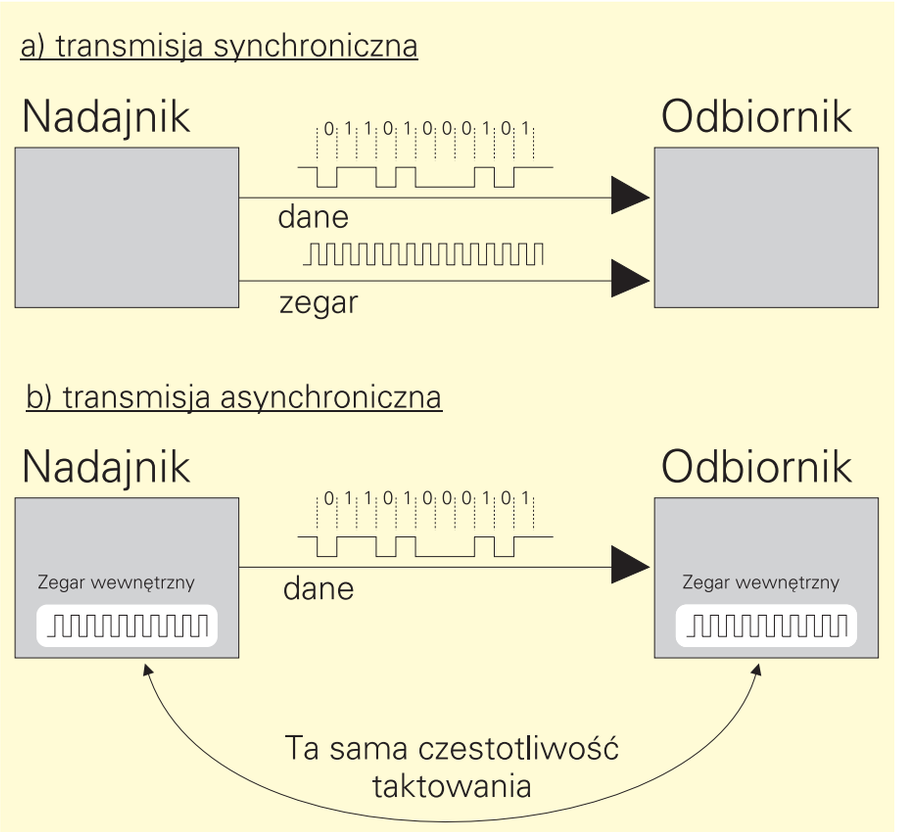
Po pierwsze, należy wiedzieć że istnieją praktycznie dwa sposoby na przysyłanie danych metodą szeregową, są to: transmisja **synchroniczna** i transmisja **asynchroniczna**.

Ponieważ port szeregowy procesora 8051 może pracować w oby tych trybach, wyjaśnię na początku o co chodzi i jakie są zasadnicze różnice pomiędzy obiema rodzajami transmisji.

Transmisja synchroniczna

W tym przypadku dane (informacje) przesyłane są od nadajnika do odbiornika za pomocą dwóch przewodów (nie licząc oczywiście masy). Jednym przesyłane są dane, a drugim generowany jest sygnał zegarowy, w takt którego odbiornik może odebrać informację i stwierdzić, czy nadeszła „1"-ka czy logiczne „0". Można więc powiedzieć że dane są przesyłane synchronicznie z przebiegiem zegarowym transmitowanym równolegle z danymi, stąd m.in. nazwa rodzaju transmisji.

Sytuację tę ilustruje rys.1a. Stan nieaktywny na linii może być umowny, może to być logiczne 0 lub jedynka, umowny jest też sposób generowania sygnału zegarowego, wszystko zależy od przyjętego w układzie rozwiązania. Nie dotyczy to konkretnie mikrokontrolera 8051 i pochodnych, lecz tu także ustalono, kiedy transmisja się zaczyna i w jaki sposób generuje się sygnał zegarowy (nazywany czasem „synchronizującym"). O tym jakie zasady obowiązują w przypadku 8051 powiem za chwilę.



Rys. 1. Transmisja synchroniczna i asynchroniczna

Jakie są zalety a jakie wady obu rodzajów transmisji?...hm, na to pytanie dość trudno jednoznacznie odpowiedzieć, zresztą jedna wyraźna korzyść od razu się rysuje:
zmniejszenie liczby przewodów w przypadku transmisji asynchronicznej.
Istnieją urządzenia które po prostu z definicji potrzebują zewnętrznego sygnału zegarowego, nie tylko do przesyłania danych pomiędzy nim a systemem nadrzędnym, lecz także do poprawnej pracy innych funkcjonalnych bloków systemu. W takich przypadkach transmisja synchroniczna staje się nieodzowna.
Do niedawna panowało mniemanie, że transmisja synchroniczna jest „bezpieczniejsza" od asynchronicznej, a także znacznie szybsza. Tak jednak było dla niedużych połączeń pomiędzy nadajnikiem a odbiornikiem (do kilkudziesięciu centymetrów dla szybkości do kilkunastu MHz). Wynikało to z tego, że jak dotąd urządzenia nadawcze i odbiorcze pracujące w trybie asynchronicznym nie były dość doskonałe, i często przy stosunkowo dużych (choć i tak małych w porównaniu z transmisją synchroniczną) prędkościach transmisji następowały „przekłamania" i „gubienie" informacji.
W dzisiejszych czasach, kiedy mamy do dyspozycji takie urządzenia jak port szeregowy w kontrolerze 8015 i podobnych, zaawansowane układy UART w komputerach PC, zdolne do transmitowania danych w trybie asynchronicznym z prędkościami nawet do 4Mb/sek. (tak !, czterech megabitów na sekundę), problem ten praktycznie zniknął. Dochodzą do tego jeszcze inne udogodnienia takie jak korekcja błędów oraz kompresja danych, co zwiększa realne szybkości transmisji oraz zwiększa bezpieczeństwo przed utratą często tak cennej informacji.

Transmisja asynchroniczna

W przypadku tego rodzaju transmisji (patrz rys.1b) nie ma oddzielnej linii zegarowej, a dane są przesyłane w takt wewnętrznego sygnału zegarowego, generowanego oddzielnie w nadajniku i odbiorniku. Warunkiem prawidłowego przesłania danych w asynchronicznym sposobie transmisji jest to, aby nadajnik i odbiornik miały ustawioną tę samą częstotliwość wspomnianych sygnałów zegarowych (nazywanych też „taktującymi"). Takie ustalenie prędkości transmisji odbywa się na różne sposoby, z reguły jest to „ręczne" ustalenie przez operatora.

Zauważcie przecież że w naszym komputerku, przed przesłaniem danych z komputera PC można ustawić (za pomocą klawisza „B"-baud) żądaną szybkość transmisji, tak samo można postąpić w portem szeregowym komputera PC, za pomocą komendy DOS'u np.:

MODE COM2: 9600, n, 8, 1 {Enter}

gdzie szybkość ustalono na 9600 bitów na sekundę (bodów). Pozostałe parametry omówię za chwilę.

Jeżeli zatem urządzenie odbiorcze uczestniczące w przesyłaniu danych wie z jaką szybkością nadawane są dane, to będzie mogło odebrać informację, bez dodatkowej linii zegarowej, jak to miało miejsce w przypadku transmisji synchronicznej. Zaoszczędzimy w ten sposób jeden „przewód". No tak, ale przecież w mikrokontrolerze 8051 są dwie linie portu szeregowego: nadawania (TXD) i odbioru (RXD). To prawda, nie jest możliwe przesyłanie danych w obie strony po jedne linii w trybie asynchronicznym (oczywiście przy wykorzystaniu sprzętowego portu szeregowego o którym mowa), lecz zauważmy że w przypadku, kiedy np. do naszego komputerka dołączymy urządzenie które potrafi tylko odbierać dane, to wystarczy połączyć je tylko 1 przewodem z końcówką TXD procesora (nie zapominając o masie), a linię RXD pozostawić nie wykorzystaną.

Z drugiej strony, wyobraź sobie że jeżeli np. do naszego komputerka dołączono oddalony o kilkadziesiąt metrów cyfrowy układ automatycznego termometru, który w określonych odstępach czasu automatycznie przesyła dane dotyczące aktualnej temperatury, to do odbioru tych danych wystarczy dołączyć linię do wejścia RXD procesora.

W obu przypadkach jeżeli chcielibyście wykorzystać transmisję synchroniczną, trzeba by pociągnąć jeszcze jeden przewód między odbiornikiem a nadajnikiem,. co zwiększyłoby koszty takiego przedsięwzięcia a jednocześnie skomplikowało instalację urządzenia.

Jednak w wielu sytuacjach fakt że procesor 8051 potrafi przesyłać danej w sposób synchroniczny, może pomóc w rozwiązywaniu wielu ciekawych zagadnień przy projektowaniu peryferyjnych układów cyfrowych, wykorzystywanych nie tylko w domowym zaciszu. Warto zatem o tym pamiętać.

UART w mikrokontrolerze 8051

Teraz, kiedy już wiecie na czym polega różnica pomiędzy transmisją synchroniczną a asynchroniczną możemy przejść do omawiania układu UART w naszym procesorze.

Oprócz rejestru SBUF istnieje dodatkowy rejestr sterujący wszystkimi funkcjami portu, a więc trybem jego pracy, sygnalizowaniem stanu transmisji, czy wreszcie uaktywnianiem odbiornika portu szeregowego. Znajduje się on pod adresem 98h w obszarze wewnętrznej pamięci danych procesora i jest jednym z rejestrów specjalnych SFR. Na ściągawce we wkładce z numeru 11/97 EdW znajduje się opis tego rejestru, jednak przedstawię poniżej nieco dokładniej znaczenie poszczególnych jego bitów.

adr. bitów	9Fh	9Eh	9Dh	9Ch	9Bh	9Ah	99h	98h	
98h	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	**SCON**

SCON.0 (RI) – (ang. „Receive Interrupt") znacznik odebrania przez port szeregowy bajtu, jest jednocześnie znacznikiem zgłoszenia przerwania (przy uaktywnionym systemie przerwań). W przypadku kiedy układ szeregowy mikrokontrolera jest ustawiony na odbiór (odblokowany jest układ odbiornika: bit REN=1), po odebraniu poprawnego znaku z urządzenia zewnętrznego, znacznik ten zostaje automatycznie ustawiony. Zerowanie tego znacznika odbywa się wyłącznie programowo, np. przez instrukcję:

CLR RI

SCON.1 (TI) – (ang."Transmit Interrupt") znacznik wysłania przez port szeregowy bajtu, jest jednocześnie znacznikiem zgłoszenia przerwania jeżeli uaktywniono wcześniej układ przerwań. W przypadku kiedy do rejestru SBUF zostanie zapisany znak (bajt) po wytransmitowaniu go przez procesor, bit ten zostaje automatycznie ustawiony (TI=1), co informuje o zakończeniu nadawania znaku przez UART. Podobnie jak w przypadku znacznika RI, znacznik ten jest ustawiany automatycznie a musi być zerowany programowo za pomocą instrukcji np.

CLR TI

SCON.2 (RB8) – (ang. „Receive Bit no. 8") port szeregowy mikrokontrolera 8051 ma możliwość odbioru i transmisji znaków 9-bitowych – istnieje specjalny tryb pracy UART, o tym za chwilę. W takim trybie w przypadku odbioru znaku z urządzenia zewnętrznego, bit RB8 zawiera właśnie wspomniany 9-ty bit odebranego znaku, Oczywiście 8 pierwszych bitów znaku znajduje się jak poprzednio w rejestrze SBUF.

SCON.3 (TB8) – (ang. „Transmit Bit no. 8") 9-ty bit nadawanego znaku w trybie transmisji z 9 bitami danych. Sytuacja analogiczna do poprzedniej, lecz w tym przypadku aby wysłać 9-bitowy znak poprzez port szeregowy należy najpierw wpisać 9-ty bajt nadawanego znaku do bitu TB8 a potem załadować rejestr SBUF ośmioma młodszymi bitami (bajtem) nadawanego znaku.

SCON.4 (REN) – („Receive ENable") bit uaktywnienia odbiornika transmisji szeregowej. W celu odbioru znaku (oczekiwania na nadejście bajtu z portu szeregowego) należy najpierw wyzerować bit REN, aby odblokować sprzętowy odbiornika znaku zawarty w mikrokontrolerze. W przypadku nadawania znaku bit ten powinien być wyzerowany (REN=0).

SCON.5 (SM2) – znacznik maskowania odbioru transmisji. Bit ten może być zmieniany programowo. Ustawienie go (SM2=1) powoduje że odbiornik ignoruje te odbierane znaki, których (w trybie 9-bitowym) 9-ty bit (RB8) jest równy zero (RB=0). W efekcie w takim przypadku nie jest ustawiany znacznik odebrania znaku (RI). Dodatkowo w trybie 8-bitowym (tryb=1) sytuacja jest identyczna kiedy po odebraniu znaku nie został wykryty bit stop'u.

Numeracja trybów ich znaczenie oraz wyjaśnienie bitów „stop'u" już za chwilę.

SCON.7 (SM0) oraz **SCON.6 (SM1)** – bity ustalające jeden z czterech trybów pracy portu szeregowego. Oto one:

SM0 SM1 = 00 – tryb 0: Transmisja szeregowa synchroniczna, znaki 8-bitowe, taktowane sygnałem zegarowym o częstotliwości Fxtal/12,

SM0 SM1 = 01 – tryb 1: Transmisja szeregowa asynchroniczna, znaki 8-bitowe, szybkość transmisji może być określana programowo (tryb do pracy np. z PC-tem)

SM0 SM1 = 10 – tryb 2: Transmisja szeregowa asynchroniczna, znaki 9-bitowe, szybkość określona jako 1/32 lub 1/64 częstotliwości zegara procesora,

SM0 SM1 = 11 – tryb 3. Transmisja szeregowa asynchroniczna, znaki 9-bitowe, szybkość transmisji może być określana programowo (znajduje także zastosowanie przy pracy z PC-tem)

Znaczenie i funkcje poszczególnych trybów są następujące.

Tryb 0

Jak już powiedziałem w tym synchronicznym trybie przesyłania informacji port szeregowy pracuje nadawając i odbierając znaki 8-bitowe. Zawsze pierwszym nadawanym lub odbieranym bitem jest najmniej znaczący (D0).

Znaki przesyłane są po dwukierunkowej linii P3.0 (RXD). Odbierane są i nadawane za pośrednictwem znanego nam już rejestru SBUF w takt sygnału zegarowego, który generowany jest przez kontroler na linii P3.1 (TXD).

W tym trybie częstotliwość sygnału zegarowego jest stała i jest równa 1/12 (jednej dwunastej) częstotliwości sygnału taktującego procesor. W przypadku użycia obwodu oscylatora procesora z rezonatorem kwarcowym 12 MHz, znaki w tym trybie będą przesyłane z szybkością 1 000 000 bitów / sek. (1 Mb/sek).

Przy nadawaniu znaku obowiązuje zasada, że zapis wysłanego kolejnego bitu znaku w urządzeniu odbiorczym (zewnętrznym np. rejestrze przesuwnym) powinien nastąpić przy **narastającym** sygnale zegarowym wytwarzanym na linii TXD.

W przypadku odbioru (REN=1) narastające zbocze sygnału zegarowego powinno powodować przesunięcie zawartości zewnętrznego rejestru przesuwnego, z którego odbierane są dane, czyli de facto odczyt odbywa się przy opadającym sygnale przesyłanym linią TXD procesora.

Po odebraniu znaku następuje automatyczne ustawienie znacznika RI, a przy nadawaniu – znacznika TI. Fakt że znaczniki te nie są zerowane automatycznie pozwala programiście na testowanie stanu ich, a co za tym idzie monitorowanie faktu odbioru czy nadania znaku bez potrzeby uruchamiania układu przerwań.

Oto praktyczny przykład ilustrujący tą właściwość.

Procedura nadania znaku z akumulatora bez korzystania z układu przerwań może wyglądać następująco:

```
OUTRS:
        mov     SBUF, A  ;przepisanie zawartości
                         ;akumulatora do rejestru UART
czekaj: jnb     TI, czekaj ;testowanie flagi nadania
                         ;(czekanie na zakończenie nadawania)
        clr     TI       ;wyzerowanie flagi nadawania
        ret              ;powrót z procedury (podprogramu)
```

Zaś procedur odebrania znaku i umieszczenia go w akumulatorze może wyglądać tak:

```
INRS:
        setb    REN              ;odblokowanie odbiornika
czekaj: jnb     RI, czekaj ;     testowanie flagi odbioru
                                 ;(czekanie na odbiór znaku)
        clr     RI               ;po odbiorze wyzerowanie flagi
        clr     REN              ;i zablokowanie odbiornika
        mov     A, SBUF          ;przepisanie znaku do akumulatora
        ret                      ;powrót z podprogramu
```

O ile podprogram nadania znaku nie zajmie z reguły więcej niż 10 cykli maszynowych procesora, o tyle sprawa odbioru znaku bez włączonego układu przerwań może być czasochłonna. Popatrzcie przecież, że w przypadku gdy nie będzie nadchodził żaden znak z portu szeregowego, procedura INRS w ogóle się nie zakończy, innymi słowy program może się „zawiesić" w przypadku gdy czekamy na jakis znak z urządzenia zewnętrznego, a on nie nadchodzi.

Właśnie dlatego użycie układu przerwań przy odbiorze znaków może okazać się bardzo pomocne. Jak tego dokonać omówię w dalszej części cyklu klasy mikroprocesorowej.

Istnieje jednak sposób zabezpieczenia się przed takim „nieskończonym" czekaniem na odbiór znaku bez angażowania często i tak przeciążonego systemu przerwań procesora. Otóż wystarczy oprócz flagi RI sprawdzać stan jakiegoś przyjętego w programie rejestru, który jest automatycznie zmniejszany o jeden np. co 1 sekundę. Załóżmy że zmniejszanie odbywa się automatycznie w procedurze przerwania generowanej przez jeden z liczników procesora np. T0. W takim przypadku, jeżeli stwierdzimy że rejestr ten (nazywany często rejestrem „przeterminowania") jest równy zero, kończymy sprawdzanie flagi RI, uznając że nastąpił błąd w odbiorze znaku. Wtedy nasza procedura odbioru znaku będzie wyglądała następująco.

Załóżmy przy tym że wspomniany rejestr „przeterminowania" nazwaliśmy np. jako „overtime" i zdefiniowaliśmy jego adres jako np.

```
overtime EQU     7Fh
```

czyli 7Fh w pamięci wewnętrznej danych kontrolera. Zakładamy też że w wywoływanej automatycznie co 1 sekundę procedurze przerwania od licznika T0 licznik „overtime" w przypadku stwierdzenia jego wartości jako różnej od zera jest zmniejszany o 1. Wtedy użycie podprogramu:

```
INRS:
        setb    REN              ;odblokowanie odbiornika
czekaj: jb      RI, jest         ;testowanie flagi odbioru
                                 ;(czekanie na odbiór znaku)
        mov     A, overtime      ;sprawdzenie rejestru
                                 „przeterminowania"
        jnz     czekaj           ;jeżeli jeszcze <>0 to czekaj
                                 na znak
        sjmp    koniec           ;jeżeli =0 to zaniechaj czekania
jest:   clr     RI               ;po odbiorze wyzerowanie flagi
        mov     A, SBUF          ;przepisanie znaku do akumulatora
koniec: clr     REN              ;i zablokowanie odbiornika
        ret                      ;powrót z podprogramu
```

z uprzednim załadowaniem rejestru „overtime" liczbą sekund przeznaczonych na maksymalne oczekiwanie na odbiór znaku poprzez instrukcję:

```
        mov     overtime, #10    ;10 sekund na odbiór znaku
        lcall   INRS             ;i wywołanie podprogramu odbioru
```

spowoduje czekanie maksymalnie 10 sekund na dobiór znaku z portu szeregowego.

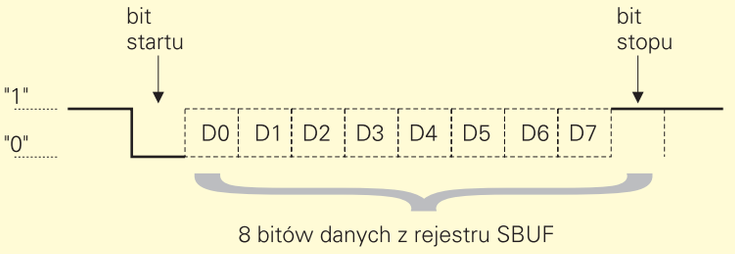
Opisana zasada znajduje zastosowanie w trzech pozostałych trybach pracy portu szeregowego, a więc w trybach 1, 2 i 3.

Tryby 1, 2 i 3

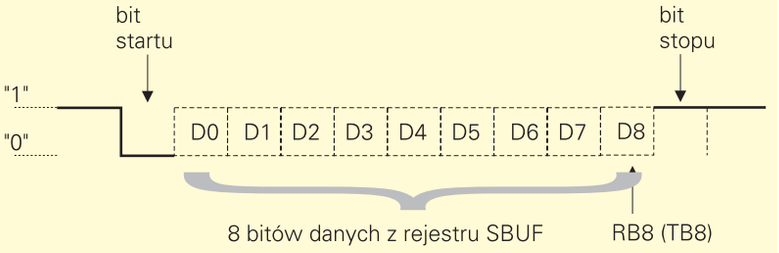
Zanim opiszę szczegółowo wspomniane trzy pozostałe tryby pracy portu szeregowego kontrolera powinienem przedstawić wspólną cechę charakteryzująca te tryby a mianowicie postać przesyłanej asynchronicznie informacji, czyli format przesyłania znaku (8-bitowy lub 9-bitowy).

Ponieważ w trybie asynchronicznym nie istnieje linia przesyłająca sygnał taktujący poszczególne nadawane i odbierane bity, obie strony nadawcza i odbiorcza muszą w jakiś sposób „wiedzieć" o tym że np. w danej chwili nadajnik rozpoczął nadawanie znaku. Wtedy odbiornik

a) format danych w trybie 8-bitowym (tryb 1)



b) format danych w trybie 9-bitowym (tryby 2, 3)



Rys.2. Format znaku w trybie 8-bitowym (a) oraz 9-bitowym (b).

detektując takie zajście będzie, znając oczywiście częstotliwość nadawani znaku przez nadajnik (znając prędkość transmisji), wiedział w jaki sposób odbierać nadawany z zewnątrz znak.

Ustalono, że podczas „ciszy na łączach", linie portów (RXD – odbioru i TXD – nadawania) są w stanie wysokim. Sygnałem rozpoczęcia nadawania znaku, a z drugiej strony sygnałem konieczności jego odbioru jest pojawienie się tzw. „bitu startu", czyli niskiego poziomu logicznego na linii (TXD w przypadku nadawania) lub (RXD – w przypadku odbioru).

Bit startu trwa dokładnie tyle ile powinny trwać (w zależności od szybkości transmisji) pozostałe bity informacji. Na **rysunku 2** oznaczono ten bit dokładnie.

Po bicie startu (pamiętajmy, zawsze równy zero!), następują kolejno bity danych. I tak pierwszy transmitowany jest najmłodszy bit (D0) bajtu wpisanego do rejestru SBUF, potem starszy (D1) i tak dalej aż do bitu D7, a w przypadku transmisji 9-bitowej dodatkowo transmitowany jest bit SCON.3 (TB8), po czym następuje bit stopu, który zawsze jest równy „1".

Pojawienie się bitu stopu kończy nadanie znaku, a po drugiej stronie jego odbiór.

Mechanizm transmisji znaku w trybach 1, 2 i 3 jest taki sam, różna jest tylko liczba bitów danych oraz szybkość transmisji, zgodnie z opisem wcześniej.

Dla przykładu powiem, że choć transmisja 9-bitowa może wydawać się mniej naturalna i niepotrzebna, to jednak jest ona często wykorzystywana do przesyłania danych z tzw. bitem parzystości. Jest to prosty sposób na wyeliminowanie odbioru zafałszowanych danych, kiedy to pomimo, że w określonej, koniecznej chwili nastąpiło wygenerowanie poprawnego bitu startu o raz bitu stopu, to jednak bity danych uległy zniekształceniu, co spowodowało zafałszowanie danych i odbiór niewłaściwego znaku.

Przy korzystaniu z transmisji z komputerem PC można poleceniem MODE ustawić tryb transmisji portu szeregowego komputera na kilka sposobów, a mianowicie:

Jak się za chwilę przekonasz nasz UART w mikrokontrolerze 8051 potrafi pracować z większymi szybkościami, lecz ustawienie tych szybkości transmisji w PC-cie jest możliwe tylko za pomocą specjalnie utworzonych programów, które są dostępne jako programu terminalowe „shareware", lub mogą być napisane przez Czytelników, tych którzy potrafią to zrobić. Najprostszym a jednocześnie ogólnie dostępnym programem twego typu jest program pochodzący z pakietu popularnego DOS-owego „Norton Commander'a" pod nazwą: „TERM90.EXE". O sposobie używania tego programu nie będę się tu rozwodził, zasady powinien znać każdy komputerowiec, a jeżeli nie to radzę się tym programem nieco pobawić.

MODE COMx: baud, parity, data, stop, retry
gdzie, poszczególne nazwy oznaczają:

x – określa numer portu szeregowego, do którego dołączony jest UART mikrokontrolera (1 dla COM1, 2 dla COM2, 3 dla COM3 itd.)
baud – określa szybkość transmisji w bitach na sekundę (standardowe najczęściej używane szybkości to: 600, 1200, 2400, 4800, 9600, 19200 bodów czyli bitów/sek)
parity – parametr określający sposób kontroli parzystości. Możliwe wartości to: „o"-odd (nieparzysty) lub „e"-even (parzysty)
data – parametr określający liczbę bitów danych. Wartości typowe akceptowane przez instrukcję MODE to 5...8
stop – liczba bitów stopu. Wartości akceptowane to 1 lub 2 (bity stopu).
Parametru „retry" nie należy w ogóle podawać.
W ten sposób można zdefiniować parametry transmisji w komputerze PC dla następujących trybów pracy portu szeregowego mikrokontrolera 8051, np.:

MODE COM2: 9600, n, 8, 1 (1)

dla trybu 1 portu UART w 8051, prędkości 9600 bodów, dołączonego do portu COM2 komputera PC, a w przypadku trybu 9-bitowego (3) :

MODE COM2: 9600, e, 8, 1 (2)

w przypadku badania parzystości bitów danych lub

MODE COM2: 9600, o, 8, 1 (3)

dla nieparzystości.

Najczęściej jednak przydaje się ustawienie portu komputera PC jak w linii (1) oraz korzystanie z trybu 1 UART kontrolera 8051.

W trybie 2 pracy, port szeregowy taktowany jest sygnałem zegarowym o częstotliwości Fxtal/32 lub Fxtal/64 (gdzie Fxtal to częstotliwość rezonatora oscylatora). O tym, która z częstotliwości będzie taktować port, decyduje stan bitu 7 (SMOD) w rejestrze SFR o nazwie PCON (adres: 87h). Ustawienie tego bitu powoduje podwojenie szybkości transmisji (Fxtal / 32) wyzerowanie – ustawienie taktowania na Fxtal/64.

Jak wspomniałem wcześniej w trybach 1 i 3 szybkość transmisji może być określana programowo. W tym przypadku układ transmisyjny taktowany jest za pomocą sygnału przepełnienia licznika T1 układu czasowo-licznikowego.

W kontrolerach 8032/52 do taktowania portu może także posłużyć licznik dodatkowy T2. Jak tego dokonać pokażę na przykładach po opisie sposobów modyfikacji i wartości szybkości transmisji.

Ponieważ wgłębianie się w przebiegi i szczegóły nadawania i odbioru znaków w trybach asynchronicznych jest mało praktyczna, zainteresowanych odsyłam do lektury [1]. Poniżej skupie się jedynie na praktycznych aspektach korzystania z dobrodziejstw portu szeregowego oraz programowania jego rejestrów.

Zanim do tego przejdę zapoznajmy się z możliwościami modyfikowania

Szybkości transmisji

W **trybie 0** szybkość przesyłania danych, jak powiedziałem wcześniej jest niezmienna i wynosi Fxtal/12.

W **trybie 2** prędkość transmisji wynosi: Fxtal/32 przy SMOD=1, lub Fxtal/64 przy SMOD=0.

W trybach 1 i 3 sprawa ma się nieco inaczej. W tym przypadku prędkość transmisji określa wzór:

$$n = \frac{Fxtal}{(256 - TH1) \times 12 \times dz}$$

gdzie: dz = 32 w przypadku gdy SMOD = 0 zaś dz = 16 gdy SMOD = 1, zaś TH1 to wartość początkowa 8-bitowego licznika TH1 (starszy bajt T1) pracującego w trybie taktowania portu szeregowego.

Aby wykorzystać standardowe prędkości transmisji znane z komputerów PC oraz innych wyposażonych w asynchroniczny port szeregowy (Commodore, Amiga), czyli: 600, 1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 76800 bit/sek należy zastosować taki kwarc (oscylator) aby po lewej stronie równania otrzymać te wartości przy pamiętaniu że wartość TH1 jest liczbą całkowitą z zakresu 0...255.

I tu wyjaśni się sprawa stosowania trochę dziwnych wartości rezonatorów kwarcowych w niektórych układach wykorzystujących procesory rodziny 8051 i ich port szeregowy.

Też to potrafisz

Ot chociażby w naszym komputerku edukacyjnym zastosowany rezonator kwarcowy ma wartość 11,0592 MHz, zobaczmy dlaczego i co w efekcie uzyskamy.

Podstawiając do powyższego wzoru, określającego prędkość transmisji danych, wartość oscylatora równą 11059200 Hz oraz zakładając że SMOD = 1 (czyli dz = 16) otrzymamy np. dla
TH1 = F4h (244 dziesiętnie): n = 11059200/((256-244)x12x16)=4800 bodów.

Postepując podobnie z innymi wartościami początkowymi licznika TH1 otrzymamy dla podanego rezonatora oraz innych (których częstotliwość rezonansowa dzieli się przez liczbę 12 i 32 (lub 16)) inne standardowe wartości, oto one:

Fxtal (MHz)	SMOD	TH1	n (bodów)
11,0592	1	FFh	57600
11,0592	1	FEh	28800
11,0592	1	FDh	19200
11,0592	1	FCh	14400
11,0592	1	FAh	9600
11,0592	1	F4h	4800
11,0592	1	E8h	2400
11,0592	1	D0h	1200
11,0592	1	A0h	600
14,7456	1	FFh	76800
14,7456	1	FEh	38400
14,7456	1	FCh	19200
14,7456	1	F8h	9600
14,7456	1	F0h	4800
14,7456	1	E0h	2400
14,7456	1	C0h	1200
1,8432	1	FFh	9600
1,8432	1	FEh	4800
1,8432	1	FDh	2400
1,8432	1	FAh	1200
1,8432	1	F4h	600

Jak zapewne się domyślacie wyzerowanie znacznika bitu SMOD w rejestrze PCON (87h) spowoduje zmniejszenie o połowę wszystkich prędkości transmisji podanych w tabeli. I tu uwaga, ponieważ rejestr PCON nie może być adresowany bitowo, czyli nie da się zmodyfikować wspomnianego bitu w tym rejestrze instrukcja np.

```
setb   SMOD
```

stąd trzeba użyć innego polecenia, a mianowicie:

```
anl        PCON, #7Fh        ;aby SMOD = 0
```

lub

```
orl        PCON, #80h        ;aby SMOD = 1
```

sprawdźcie na kartce papieru lub kalkulatorze.

W tabeli podano najczęściej używane wartości rezonatorów kwarcowych dla zastosowań portu szeregowego 8051. Najbardziej popularną jest wartość 11,0592 MHz, dość rozpowszechniona w handlu, w szczególnych zastosowaniach przydaje się rezonator o Fxtal = 14,7456 MHz, kiedy chcemy aby nasz mikroprocesor konsumował jak najmniej prądu a jednocześnie pracował przy kilku standardowych prędkościach transmisji, powinniśmy użyć kostki w wersji CMOS (80C51/80C31) oraz zastosować rezonator o Fxtal = 1,843200 MHz.

Możliwe jest także w praktyce stosowanie „okrągłych" wartości rezonatorów kwarcowych, lecz w tym przypadku należy liczyć się z błędami szybkości, które jednak przy małych prędkościach transmisji nie mają wielkiego znaczenia dla poprawnej pracy układu UART mikrokontrolera 8051. Wynika to z faktu że błędnie odebrany bit zawsze „leży" poza transmitowanymi 9-cioma czy 10-cioma bitami (nie występuje), które wchodzą w skład transmitowanego w trybie asynchronicznym słowa. I tak np. z zastosowaniem rezonatora kwarcowego Fxtal = 12 MHz możemy śmiało wymieniać dane z komputerem PC lub innym urządzeniem wykorzystującym łącze szeregowe i standardowe prędkości transmisji prędkościami:

2400 bodów (TH1 = F3h, SMOD = 0), wtedy błąd wyniesie tylko: 0,16%
4800 bodów (TH1 = E6h, SMOD = 0), wtedy błąd wyniesie także 0,16%

ale już przy prędkości transmisji równej
9600 bodów (TH1 = F9h, SMOD = 1), błąd wyniesie aż 7%, bo aktualna szybkość transmisji będzie mniejsza od założonej i wyniesie w tym przypadku: 8923 body.

W przypadku użycia do taktowania portu szeregowego w kontrolerach 8032/52 licznika T2, prędkość transmisji określona będzie wzorem:

$$n = \frac{Fxtal}{(65536 - RLD) \times 16 \times 2}$$

gdzie RLD to wartość początkowa, zapisana przez program użytkownika do pary rejestrów RLDH i RLDL. Jeżeli zastosujemy rezonator kwarcowy o częstotliwości np. 16 MHz to możliwe do uzyskania częstotliwości transmisji będą z przedziału:
7...50000 bitów / sek.

Praktyczne wskazówki

Jak zatem zaprogramować rejestr sterujący portu szeregowego oraz jak zmusić do pracy licznik którego zadaniem będzie taktowanie transmisji. Oto przykład który pomogą wam zrozumieć ten problem.

Przykład

Zaprogramujemy UART tak, że będzie pracował w trybie asynchronicznym 8-bitowymi znakami, a taktowany będzie licznikiem T1. Prędkość transmisji określimy np. na 19200 bitów / sek (bodów). Inne wartości transmisji – patrz tabela.

Aha, byłby zapomniał, zakładamy że w systemie zastosowano rezonator kwarcowy o Fxtal = 11,0592 MHz.

Oto sekwencja początkowa uruchamiająca pracę portu:

```
INIT_RS:
   mov     SCON, #01000000b        ;tryb 1 portu
   orl     TMOD, #00100000b        ;licznik T1 tryb pracy 8-bitowej
                                   ;liczy TL1 z automatycznym
                                   ;wpisywaniem wartości
                                   ;początkowej
                                   ;z TH1
   orl     PCON, #80h              ;ustawienie SMOD = 1
   mov     TH1, #FDh               ;prędkość = 19200 bodów
                                   ;(tabela)
   setb    TR1                     ;start licznika T1 (TL1)
   .....                           ;i gotowe...
```

Teraz wystarczy użyć procedur które opisałem wcześniej (INRS i OUTRS) aby wysłać dowolny znak poprzez łącze szeregowe. Zmieniając wartość ładowaną do rejestru TH1 (linia: mov TH1, #xx) można w dowolnej chwili zmieniać prędkość transmisji w zakresie zależnym oczywiście od zastosowanego rezonatora kwarcowego.

Analogicznie, opierając się o opis rejestru licznika T2CON (w kostkach 8032/52) można zaprogramować i zmusić do pracy licznik T2, radzę poeksperymentować w domu, oczywiście zaopatrując się wcześniej w odpowiedni mikrokontroler.

No dobrze ale czy można bezpośrednio dołączyć końcówki portu procesora 8051 (TXD i RXD) do np. komputera PC ?... absolutnie nie. Potrzebny jest do tego odpowiedni konwerter poziomów napięć. Przykład takiego rozwiązania powinien nasunąć się Wam sam, no tak przecież w naszym komputerku edukacyjnym zastosowaliśmy taki właśnie układ, wykorzystujący popularną przetwornicę w postaci układu scalonego MAX232 lub ICL232.

Więcej na temat portu szeregowego możecie przeczytać w kilku ostatnich numerach EdW (6, 7, 9 z roku 1997).

Dodatkowe szczegółowe informacje oraz specyficzne cechy układu UART mikrokontrolera 8051 o pochodnych znajdziecie w dodatkowej literaturze [1].

Sławomir Surowiński

Literatura

[1] A. Rydzewski – Mikrokomputery jednoukładowe rodziny MCS-51, WNT1992, 1995

Lekcja 7

W dzisiejszej lekcji rozwiążę zadania z poprzedniego miesiąca i wyjaśnię sposób wykonywania przytoczonych procedur arytmetycznych przez mikrokontroler 8051. dla ułatwienia będę numerował wszystkie linie w nawiasach tuż obok komentarza.

1. Procedura dodawania dwóch liczb 16-bitowych

Pierwszy składnik będzie np. umieszczony w rejestrach DPH i DPL (DPTR)

Drugi składnik dodawania będzie umieszczony w rejestrach B (starszy bajt) i A (młodszy bajt). W wyniku dodawania powstanie liczba w rejestrach B.A oraz dodatkowo przy przekroczeniu zakresu liczb zostanie ustawiony znacznik przeniesienia C.

```
; C.B.A = DPTR + B.A
ADD16:
   add    A, DPL   ;(1) dodanie LSB
   push   Acc      ;(2) przechowanie LSB wyniku na stosie
   mov    A, B     ;(3) przepisanie składnika MSB do Acc
   addc   A, DPH   ;(4) dodanie składnika 2
   mov    B, A     ;(5) przepisanie wyniku do rejestru B
   pop    Acc      ;(6) odtworzenie LSB wyniku
   ret             ;(7) i koniec (powrót) podprogramu
```

W podprogramie wykonano operację:

```
          B       A
+         DPH     DPL
-------------------------
C         B       A
```

Popatrzmy na zapis matematyczny. Najpierw dodajemy składniki a prawej strony, czyli A i DPL (linia 1). Potem aby dodać starsze bajty (B i DPH) trzeba było użyć znowu akumulatora, lecz aby to zrobić należało najpierw zachować wynik z Acc, stąd linia (2) – posłużono się stosem. Następnie przepisano zawartość B do A (linia 3), aby ją dodać za chwilę (linia 4) do MSB drugiego składnika, który przecież znajduje się w DPH. Na końcu przepisano tak powstałą sumę MSB do rejestru B (linia 5) i odtworzono akumulator (linia 6).

A teraz dodatkowe zadanie, poniżej przedstawię alternatywny zapis powyższej procedury, spróbuj się zastanowić, dlaczego oba przykłady w efekcie wykonują to samo?

```
; C.B.A = DPTR + B.A
ADD16_2:
        add     A, DPL
        xch     A, B
        addc    A, DPH
        xch     A, B
        ret
```

2. Procedura odejmowania dwóch liczb 16-bitowych

Odejmowanie dwóch liczb 16-bitowych przeprowadzimy w podobny sposób jak to miało miejsce z dodawaniem, a więc od pary rejestrów B.A odejmiemy zawartość DPTR, wynik pozostanie w B.A

```
; B.A = B.A – DPTR
SUBB16:
        clr     C               ;wyzerowanie znacznika C
        subb    A, DPL
        push    Acc
        mov     A, B
        subb    A, DPH
        mov     B, A
        pop     Acc
        ret
```

prawda że proste! komentarz jest chyba zbędny, poza wyjaśnieniem, dlaczego na początku procedury wyzerowano znacznik przeniesienia C. Otóż instrukcja SUBB procesora 8051 powoduje z definicji odjęcie dwóch składników działania z uwzględnieniem znacznika C. Toteż profilaktycznie należy go wyzerować przez wykonaniem procedury, inaczej w przypadku gdy będzie C=1, wynik będzie obarczony błędem (będzie za mały o 1). Podobnie jak poprzednio procedurę można zapisać jako:

```
; B.A = B.A – DPTR
SUBB16_2:
        clr     C
        subb    A, DPL
        xch     A, B
        subb    A, DPH
        xch     A, B
        ret
```

Uwaga! W przypadku kiedy B. A będzie mniejsze od DPTR, obliczony zostanie moduł różnicy, a ujemny znak działania będzie sygnalizowany ustawieniem znacznika C!

3. Procedura mnożenia liczby 16-bitowej przez 8-bitową

W tym przypadku pomnożymy zawartość rejestru DPTR przez Acc (akumulator). Wynik będzie zatem 24-bitowy, a umieszczony w DPTR. A (DPH. DPL. A)

Zauważmy że mnożenie możemy zapisać jako:

DPH. DPL x Acc = DPL x Acc + DPH x Acc x 100h

Jak widać najpierw trzeba wymnożyć starszy (MSB) oraz młodszy bajt (LSB) rejestru DPTR przez akumulator, a potem odpowiednio dodać składniki. Wykonam zatem działania:

wymnożę DPL x Acc
mnożę DPH x Acc
a następnie dodam w kolejności

```
          DPH     DPL     00h
+                 B       A
-------------------------------
          DPH     DPL     A
```

i w ten sposób otrzymam wynik. Skorzystam przy tym z instrukcji mnożenia dwóch liczb 8-bitowych. A oto procedura:

```
MUL24:
        push    Acc     ;przechowanie mnożnika na stosie
        mov     B, DPH
        mul     AB      ;mnożenie DPH x Acc
        mov     DPH, B  ;i zapisanie wyniku – MSB
        push    DPL     ;przechowanie wyniku pośredniego
        mov     DPL, A  ;przepisanie LSB wyniku mnożenia DPH x Acc
        pop     B       ;odtworzenie wyniku pośredniego
        pop     Acc     ;odtworzenie mnożnika
        mul     AB      ;mnożenie DPL x Acc
        xch     A, B
        add     A, DPL  ;dodanie Acc + DPL
        mov     DPL, A
        jnc     less    ;czy > FFh
        inc     DPH     ;tak to korekcja na DPH wyniku
less:   xch     A, B    ;odtworzenie Acc
        ret             ;...i koniec mnożenia, wynik w DPTR. A
```

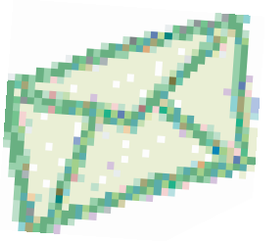
4. Procedura dekrementacji z korekcją akumulatora
Jak wiemy zastosowanie instrukcji

```
dec     A
```

w przypadku kiedy w akumulatorze znajduje się liczba np. 50h da w rezultacie wynik w postaci liczby 4Fh. A co zrobić aby zamiast takiego wyniku otrzymać po „podobną do ludzi" liczbę 49h. Do takiej dekrementacji czyli zmniejszania o 1 liczb zapisanych w kodzie BCD, posłużyć może poniższa procedura, oto ona:

```
DECACC:
        jnz     nie0        ;(1) sprawdzenie czy Acc=0
        mov     A,#99h      ;(2) tak to załadowanie liczby 99h
        ret                 ;(3) i powrót z podprogramu
nie0:   clr     C           ;(4) nie to zeruj wskaźnik C
        subb    A,#1        ;(5) odejmij od akumulatora 1
        push    Acc         ;(6) i przechowaj wynik na stosie
        anl     A,#0Fh      ;(7) zamaskuj 4 msb wyniku
        cjne    A,#0Fh,nieF ;(8) czy druga cyfra to „F"?
        pop     Acc         ;(9) tak to odtwórz wynik
        anl     A,#0F0h     ;(10) i zamaskuj 4 lsb
        orl     A,#9        ;(11) a na miejsce 4 lsb wpisz cyfrę „9"
        ret                 ;(12) i zakończ podprogram
nieF:   pop     Acc         ;(13) jeżeli druga cyfra to nie „F" to
                            ;odtwórz wynik
        ret                 ;(14) i zakończ podprogram
```

Procedura sprawdza trzy warunki:
a) jeżeli w akumulatorze jest liczba 0, to wpisywana jest liczba 99h i procedura zostaje zakończona (linie 1...3)
b) jeżeli po zwyczajnym odjęciu jedynki (linia 5) druga cyfra wyniku jest równa „F" to przeprowadzana jest konwersja wyniku do postaci BCD – linie 9...11. Taka sytuacja ma miejsce gdy liczba wejściowa jest „okrągła" czyli np. 30h, 40h, 50h, 60h, itp.
c) jeżeli po odjęciu jedynki druga cyfra wyniku jest z przedziału 0...9 to nie rób nic tylko opuść podprogram – wynik jest w porządku. Sytuacja taka mam miejsce w przypadku gdy liczba wejściowa ma na drugiej pozycji cyfrę z przedziału: 1...9.



Kącik pocztowy 8051

Zasypany wieloma listami, dotyczącymi cyklu artykułów poświęconych programowaniu mikrokontrolerów 8051, postanowiłem uruchomić kącik pytań i odpowiedzi, które kierujecie do mnie w swoich listach.
Na wstępie chciałem bardzo podziękować za każdy list, zarówno te pochwalne jak i krytyczne. Przyznam że, cieszy mnie bardzo fakt, iż tak wielu z Was zdecydowało się sięgnąć po „mikroprocesory", a co najważniejsze odnosi już małe ale jak ważne w nauce sukcesy.

Na początek chciałem szczególnie podziękować: **Rafałowi Majdzie** z Krakowa, **Hubertowi Hawłasowi** z Warszawy, **Zdzisławowi Chmielewskiemu** z Tomaszowa Mazowieckiego, **Robertowi Szymaszkowi** z Bielsko-Białej, **Tomaszowi Jabłonowskiemu** z Moniek, **Jarosławowi Chudobie** z Gorzowa Wlkp.

Specjalnie podziękowania także dla pana **Wiesława Zacharskiego** z Karoliną i Piotrkiem.

Chciałbym jednocześnie uspokoić **Konrada Iwaniuka** z Mysłowic, drogi Konradzie prześlij swój komputerek pod adresem redakcji AVT z dopiskiem na kopercie „Serwis", problem zostanie rozwiązany.

Wyrazy uznania dla **Przemka Pietrzyka** z Lublina za jego wyczerpujący list z zamieszczonymi, własnoręcznie napisanymi programikami – brawa „ręczniakom".

Gorąca prośba ode mnie: piszcie proszę swoje adresy nie tylko na kopertach ale także w treści korespondencji. Przyznam że często koperty z natury swojej zostają oddzielone o listu i późniejsze dopasowanie obu części jest praktycznie niemożliwe.
No dobrze, przejdźmy zatem do kilku listów.

*

...Jestem początkującym studentem telekomunikacji jak również czytelnikiem gazety EdW. ... Nawiązując do sprawy z zaciekawieniem śledzę artykuły dotyczące 8051, lecz zakup kitu jest dość kosztowny i moje pytanie i prośba do Was – czy jest możliwość zakupienia samej kostki EPROM wraz z „monitorem"?

Rafał Majda z Krakowa

Odpowiedź: No cóż, nie wszystkim dobrze się powodzi, aczkolwiek przyznam że trochę jestem zaskoczony twierdzeniem że kit komputerka jest kosztowny. Muszą przyznać że AVT poczyniło wszelkie kroki, aby ów zestaw był jak najtańszy. Porównując obecne ceny układów scalonych oraz płytek drukowanych doszedłem wielokrotnie do wniosku, że samodzielny zakup elementów oraz wykonanie (ręczne !, tak ręczne) płytek w sumie daje koszt zbliżony do ceny zestawu. Dochodzi do tego sprawa pozornej prostoty układu, i jak wynika z waszych listów, częste próby wykonywania układu na własnych PCB kończą się problemami. Podziwiam jednak determinację Czytelników i chęć wykonania wszystkiego samemu (sam taki byłem), dlatego informuję że zaprogramowany EPROM można nabyć za pośrednictwem Działu Handlowego AVT.

*

...Jeżeli chodzi o sam układ (elektroniczny) to zastrzeżeń nie mam, ale jeżeli chodzi o zaprojektowana płytkę, to zastanawia mnie kilka rzeczy: czy umieszczone złącze ARK do zasilania jest tu najlepszym rozwiązaniem... lepiej by było zastosować standardowy wtyk zasilania. nie wiem czym kierował się projektant płytki podłączając w dość dziwnej kolejności linie do złącza PORT, nie wierzę w to że na dwustronnej płytce nie dało się połączyć tego normalnie. Jak już wspomniałem mam płytki przykręcone jedna nad drugą i prawdę mówiąc jedne otwór mi nie pasował, wiem że przeszkadza wyświetlacz, ale czy nie dało się tego poprzesuwać...

Hubert Hawłas z Warszawy

Jako odpowiedź cytuje fragment innego listu **Andrzeja Milewskiego z Torunia:**

... trafne moim zdaniem było zastosowanie typowego złącza ARK roli gniazda doprowadzającego zasilanie do układu. W wielu konstrukcjach – kitach AVT używacie nietypowych, różnych gniazd zasilających, a każdy elektronik zdaje sobie sprawę, że w tej dobie na rynku nie panuje praktycznie żaden standard, przynajmniej jeżeli chodzi o układy niskonapięciowe...

Sposób wyprowadzania sygnałów na złącz komputerka opisuje **Krzysztof Wójcik** z Gdańska, mianowicie:

... zresztą każdy wie, że porządkowanie sygnałów na złączach powoduje zwiększenie komplikacji druku a im więcej przelotek na płytce szczególnie dwustronnej, tym większe prawdopodobieństwo uszkodzenia mechanicznego płytki podczas wielu godzin pracy z komputerem edukacyjnym.... I chwała za to autorowi, bowiem przyznać musze że wykonując płytki samodzielnie miałem mniej otworów do wiercenia...

Sprawa sposobu mocowania obu płytek, obok siebie, czy jedna nad drugą, jest dyskusyjna, w każdym razie otrzymuję od Czytelników zdjęcia komputerków obudowanych w różny sposób. Wszystkie wyglądają wspaniale! Oby tak dalej.

Nieco szerszy list napisał **Przemysław Piertrzyk** z Lublina, który postanowił zastosować w czasie świąt komputerek do sterowania lampkami na choince – i bardzo dobrze!

... Wykorzystałem do tego specjalne złącze z wyprowadzeniami portu P1 i na próbę zmontowałem układ z 8-miu diod LED tak by każda odpowiadała innemu bitowi portu. Układ próbny miałem gotowy, teraz wystarczył tylko program. ...

Ponieważ Przemek jest ręczniakiem, musiał samodzielnie napisać a następnie zamienić na kod maszynowy programik do sterowania lampkami. Pierwsza wersja programu nie działała poprawnie – powodowała zawieszanie się systemu, druga zaś jak pisze nasz czytelnik działa znakomicie, dlatego przytaczam ja poniżej jako dobry i prosty przykład. Przemek do efektu tzw. biegnącego światła wykorzystał instrukcję rotacji akumulatora (RL A i RR A). Oto listing.

```
kod          etyk   mnemonik        komentarz
74FE         pr1:   mov  A,#254     ;załadowanie do A liczby 254
                                    (11111110)
12801A       pr_1:  lcall wait      ;wywołanie podprogramu
                                    ;opóźnienia
F590                mov  90h, A    ;wyświetlenie A w porcie P1
23                  rl   A         ;rotacja bitów A w lewo
                                    ;(11111101)
309702              jnb  97h, pr2  ;skok jeśli najstarszy bit P1
                                    jest =0
80F5                sjmp pr_1      ;skok do etykiety pr_1
747F         pr2:   mov  A, #127   ;załadowanie do A liczby 127
                                    ;(01111111)
12801A       pr_2:  lcall wait      ;wywołanie programu
                                    ;opóźnienia
F590                mov  90h, A    ;wyświetlenie A w porcie P1
03                  rr   A         ;rotacja bitów A w prawo
3090E1              jnb  90h, pr1  ;skok do pr.1 jeżeli najmłodszy
                                    ;bit P1=0
                                    ;następny cykl
80F5                sjmp pr_2      ;skok do etykiety pr_2
                                    ;(od początku)

             wait:                  ;definicja procedury „wait"
C0E0                push Acc       ;przechowaj A na stosie
7433                mov  A, #51    ;czas opóźnienia = 51 x 4 ms
                                    ;(ok. 200ms)
120295              lcall DELAY    ;wywołanie systemowej
  proc.                             opóźnienia
D0E0                pop  Acc       ;odtworzenie akumulatora
22                  ret            ;i powrót z podprogramu
```

Przemysław pyta także o możliwość zastosowania komputerka jako timera- zegara, tematem tym zajmę się w kolejnym odcinku przy okazji omawiania układu przerwań kontrolera i układu licznikowego oraz sposobów praktycznego ich programowania.

I na koniec uwaga Przemka dotycząca treści artykułu klasy mikroprocesorowej z 11 numeru EdW z poprzedniego roku, gdzie objaśniano przykład obliczania skoku do etykiety „w górę". W artykule wkradł się błąd, napisano mianowicie, że:

... jeżeli policzona ilość bajtów jest równa 10 (0Ah), to przy skoku „w górę będzie:

U2ofset = **100h** – F6h..."

Oczywiście zamiast liczby 100h powinno być FFh, za zwrócenie uwagi na błąd w treści bardzo dziękuję, jak widać chochlik drukarski nie śpi.

*

Tomasz Jabłonowski jako „komputerowiec" w swoim liście porusza sprawę sposobu pisania programów źródłowych tak, aby były akceptowane przez kompilator oferowany na dyskietce : AVT-2250/D. Otóż aby wszystko było w porządku, jak już wspominałem przy pisaniu programów, obowiązuje zasada, że wszystkie etykiety powinny zaczynać się od pierwszej kolumny tekstu (czyli najbliżej brzegu). Pan Tomasz zwraca uwagę, że w artykułach klasy mikroprocesorowej przy opisywaniu kompilatora AVT-2250/D nie podano tego faktu.

Odpowiadam: na dyskietce znajduje się zbiór tekstowy, w którym wymienione są wszystkie zasady pisania programów źródłowych na zawarty kompilator, proszę tylko go przeczytać. W artykułach pominięto niektóre informacje zawarte w treści tego zbioru, ze względu na fakt, że część czytelników tzw. „ręczniaków" nie korzysta w ogóle z kompilatora, i taki opis byłby praktycznie zbędny.

Dlatego zachęcam do przestudiowania zbioru tekstowego zawartego na dyskietce, z pewnością rozwieje on wszystkie wątpliwości.

W swoim jakże ciekawym liście, Tomasz porusza także sprawę prób wyświetlania liczb większych niż te akceptowane przez procedurę A2HEX, opisywaną przy okazji prezentowania Bios'u komputerka. Przytacza prosty programik próbny, dzięki któremu na pozycjach DL3,4, i 5 zostają wyświetlona liczba 123.

Zamieszcza listing tego programu, oto on:

```
include  const.inc
include  bios.inc

         org     8000h
         lcall   CLS
         mov     DL3, #_1
         mov     DL4, #_2
         mov     DL5, #_3

         END
```

Program jest jak najbardziej poprawny, z jednym wyjątkiem, jego wpisanie do komputerka i wykonanie spowoduje że system „pójdzie w maliny", a to ze względu na brak na końcu programu przed deklaracją END pętli zatrzymującej program w postaci np.:

```
stop:          SJMP    stop
```

Ponieważ mówiłem w artykule dwa miesiące temu o konieczności stosowania tego polecenia, nie będę przypominał konsekwencji wykonania programu, który kończy się bez tej nieskończonej pętli (lub innego skoku bezwzględnego „gdzieś w górę" programu). Proszę zatem pamiętać o stosowaniu tego zalecenia, w przeciwnym przypadku przy wykonaniu tego programu nie zobaczy się efektu swojej pracy, bowiem natychmiast po wyświetleniu liczby 123, na wyświetlaczu prawdopodobnie pojawi się komunikat „-HELLO", znany wszystkim, a sygnalizujący zresetowanie komputerka.

W kolejnej skrzynce porad omówię pozostałe listy, na razie pozdrawiam wszystkich sympatyków i uczestników Klasy Mikroprocesorowej!

Sławomir Surowiński