Mikrokontrolery? To takie proste...

Część 11
Asembler – język maszynowy procesora

W kolejnej części naszego cyklu zapoznamy się z pozostałymi podprogramami zawartymi w monitorze komputerka edukacyjnego AVT-2250. W dalszej części artykułu postaram się wyjaśnić znaczenie kilku ważnych dla działania monitora rejestrów. Na końcu jak zwykle proponuję kolejną lekcję szkoły mikroprocesorowej, tematem dziesiejszej będzie konwersacja liczb szesnastkowych na dziesiętne.

Na początek naszego dzisiejszego spotkania kilka użytecznych przy pisaniu programów podprocedur standardowych „biosu" naszego komputerka.

TEXT (0285h)

– wyprowadza na wyświetlacz znaki począwszy od pozycji DLx podanej w rejestrze B aż do kodu „pustego" – 00h. Kody muszą znajdować się w pamięci programu, a adres ich początku przed wywołaniem procedury powinien znajdować się w rejestrze DPTR. Jeżeli kolejne kody (bajty) opisują znaki np. cyfry, litery: „A""...""F", „P", „L", oraz inne zgodnie z zasadą tworzenia własnego znaku podaną w poprzednim odcinku szkoły mikroprocesorowej, a cały ciąg bajt kończy się liczbą 0, to w efekcie podprogram powoduje wypisanie na wyświetlaczu pseudo-tekstu.
– adres wywołania: 0285h
– we: DPTR wskazuje początek ciągu bajtów w pamięci programu, B – pozycja 1 znaku na wyświetlaczu (1...8)
– wy: wypisuje znaki na wyświetlaczu zgodnie z zasada opisaną wcześniej
– traci: rejestry DPTR, A i R0
– przykład: powiedzmy że chcemy wypisać pseudo-napis informujący o błędzie, np. „Error". W pierwszej kolejności dobrze jest zdefiniować trzy znaki (litery), a więc „E", „r" i „o" opierając się na matrycy wyświetlacza 7-segmentowego. Korzystając z zasady tworzenia znaków opisanej w lekcji 5, definiujemy :

```
_Eequ   01111001b ;litera „E" (segmenty B, C i kropka zgaszone) – 79h
_r equ  01010000b ;litera podobna do małego „r" – 50h
_oequ   01011100b ;litera podobna do małego – 5Ch
```

Przy definicji każdej z liter dodano znak podkreślenia, ze względu na komputerowców, którzy korzystając z asemblera nie powinni tworzyć (z zasady) jednoznakowych przypisań EQU. Ja w swoich rozważaniach przyjąłem zasadę, że przy definiowaniu znaku do wyświetlenia poprzedzam jego symbol właśnie znakiem określenia, a np. definiując kod wciśniętego klawisza dodaję na początku literę „k", ot tak dla porządku i większej czytelności całego programu.

No dobrze ale wracajmy do naszego przykładu, otóż teraz w dowolnej części programu, najlepiej na jego końcu (przed deklaracją END) należy zdefiniować cały napis, a więc:

```
napis_blad          DB      _E, _r, _r, _o, _r, 0
```

Prawda że proste, zauważ wszakże że do prawidłowego wyświetlenia potrzebny jest kod 0 (00h) na końcu każdego ciągu bajtów i tak też jest u w naszym przykładzie.

Teraz ilekroć będziesz chciał wywołać taki komunikat w swoim programie zamiast wystarczą te oto instrukcje:

```
MOV     DPTR, #napis_blad   ;adres ciągu znaków (bajtów)
MOV     B, #1               ;wypisz od 1-szej pozycji displeja
LCALL   TEXT                ;no i wypisz komunikat
```

„Komputerowcy" mogą w tej chwili zajrzeć do zbioru o nazwie „CONST.INC" z dyskietki AVT-2250/D, gdzie zdefiniowano większość potrzebnych znaków jednocześnie na tyle czytelnych aby być rozpoznane na wyświetlaczu wskaźnika 7-segmentowego. „Ręczniakom" podaję ściągę, oto ona:

```
znak            kod             komentarz
_0      equ     00111111b       ;cyfra '0' (3Fh)
_1      equ     00000110b       ;cyfra '1' (06h)
_2      equ     01011011b       ;cyfra '2' (5Bh)
_3      equ     01001111b       ;cyfra '3' (4Fh)
_4      equ     01100110b       ;cyfra '4' (66h)
_5      equ     01101101b       ;cyfra '5' (6Dh)
_6      equ     01111101b       ;cyfra '6' (7Dh)
_7      equ     00000111b       ;cyfra '7' (07h)
_8      equ     01111111b       ;cyfra '8' (7Fh)
_9      equ     01101111b       ;cyfra '9' (6Fh)
_A      equ     01110111b       ;litera 'A' (77h)
_B      equ     01111100b       ;litera 'B' (7Ch)
_C      equ     00111001b       ;litera 'C' (39h)
_D      equ     01011110b       ;litera 'D' (5Eh)
_E      equ     01111001b       ;litera 'E' (79h)
_F      equ     01110001b       ;litera 'F' (71h)
_kropka equ     10000000b       ;kropka wyświetlacza (80h)
_pusty  equ     0               ;wszystkie segmenty
                                wygaszone (00h)
_kursor equ     00001000b       ;znak podkreślenia „_", znany ja-
ko kursor
_minus  equ     01000000b       ;znak '-' (40h)
_H      equ     01110110b       ;litera 'H' (76h)
_J      equ     00011110b       ;litera 'J' (1Eh)
_L      equ     00111000b       ;litera 'L' (38h)
_P      equ     01110011b       ;litera 'P' (73h)
```

```
_U      equ     00111110b   ;litera 'U' (3Eh)
_Y      equ     01101110b   ;litera 'Y' (6Eh)
_M      equ     00110111b   ;znak podobny do dużej litery 'M' (37h)
_n      equ     01010100b   ;znak podobny do małej litery 'n' (54h)
_T      equ     00110001b   ;znak podobny do dużej litery 'T' (31h)
_S      equ     _5          ;litera „S" jest taka sama jak cyfra „5"
_r      equ     01010000b   ;znak podobny do małego 'r' (50h)
```

W nawiasach podano, użyteczne przy ręcznym wklepywaniu programów, szesnastkowe postacie kodów przedstawionych znaków. Oczywiście podane przykłady nie wyczerpują możliwości definiowania własnych znaków i symboli, przecież segmentów w wyświetlaczu jest 7, czyli teoretycznych kombinacji aż 128, do tego dochodzi jeszcze „kropka", ale to pozostawiam już waszej wyobraźni oraz naszym dalszym wspólnym rozważaniom.

DELAY (0295h)

– wykonanie tego podprogramu zajmuje procesorowi określony czas, którego wielkości można określić podając wartość rejestru A (akumulatora przed wywołaniem procedur, zgodnie z zasadą:
*czas wykonania = wartość z Acc * 1,95 ms (milisekundy) – około.*

Skoro więc do akumulatora można wpisać dowolną liczbę z zakresu 1...255 (nie zalecam w tym przypadku zera) to w efekcie działanie podprogramu DELAY można porównać z wywołaniem celowego opóźnienia o czasie trwania w zakresie około 2...500 ms, czyli od ułamków do prawie do 1/2 sekundy.

– adres wywołania: 0295h
– we: Acc – opóźnienie * 1,95 ms
– wy: j/w
– traci: Acc
– przykład:

```
MOV     A, #255
LCALL   DELAY           ;około 1/2 sekundy
MOV     A, #255
LCALL   DELAY           ; i znów około 1/2 sekundy
.........
```

co w programie wywoła opóźnienie około 1 sekundy.

Uwaga, procedury nie należy wykorzystywać w celu generowania opóźnień czasowych, np. przy odliczaniu sekund, minut czy godzin, ponieważ jej konstrukcja powoduje błąd odmierzania. Do tego celu nadaje się inny rejestr w pamięci wewnętrznej RAM procesora, ale o tym za chwilę.

CONIN (02C5h)

– zeruje bufor klawiatury (komórkę w wewn. RAM o adresie 76h) a następnie oczekuje aż do skutku na wciśnięcie dowolnego klawisza, po czym umieszcza w akumulatorze (Acc) kod tego klawisza zgodnie z tabelą ASCII, czyli

przy wciśnięciu klawisza:	w Acc będzie kod:
"0"	30h
"1"	31h
"2"	32h
"3"	33h
"4"	34h
"5"	35h
"6"	36h
"7"	37h
"8"	38h
"9"	39h
"A"	41h
"B"	42h
"C"	43h
"D"	44h
"E"	45h
"F"	46h

Wyjątkiem jest klawisz OK, w wyniku naciśnięcia którego w akumulatorze znajdzie się kod 0Dh (13 dziesiętnie).

Ktoś w tej chwili zapyta a co się stanie w przypadku wciśnięcia klawisza „M", a no nic ponieważ ten klawisz zarezerwowany jest do natychmiastowego przerwania programu użytkownika i powrotu do monitora komputerka edukacyjnego.

– adres wywołania: 02C5h
– we: czekanie na klawisz aż do skutku
– wy: w Acc kod naciśniętego klawisza zgodnie z tabelą powyżej
– przykład: powiedzmy że czekamy na klawisz „OK" lub na „4", poniższy przykład pozwoli nam rozstrzygnąć, który z nich został wciśnięty. Zdefiniujemy na początku kod klawisza OK jako:

```
klaw_OK         EQU             13

czekaj:
        LCALL   CONIN           ;oczekiwanie na klawisz
        CJNE    A, #klaw_OK, czy4   ;czy klawisz = OK, nie to skocz
                                jest_OK:
        .......     tu dalsze instrukcje programu właściwe dla klawisza OK
        .......
czy4:   CJNE    A, # 4, czekaj      ;czy klawisz '4, nie to czekaj dalej
        .......
        .......     tu dalsze instrukcje programu właściwe dla klawisza '4
```

Zauważmy, że w linii sprawdzającej klawisz '4 użyliśmy znaków apostrofów, dlaczego?, ano dlatego że argumentem bezpośrednim w przypadku asemblera PASM51.EXE nie musi być bezpośrednio liczba, ale także może być odpowiadający jej kod znaku zgodny z ASCII, jak podano w tabeli powyżej. Dzięki temu program staje się bardziej czytelny, a komputerowcy nie musza sobie wyrywać włosów z głowy, zadając sakramentalne pytanie podczas analizy dawno zapomnianego programu: „... co ja w tym momencie chciałem osiągnąć???.....".

Linię tę można oczywiście zapisać jako

```
czy4:   CJNE            A, #34h, czekaj         ;.... itd.
```

a efekt działania będzie ten sam. Należy przy tym wspomnieć że ta forma jest bardziej czytelna dla „ręczniaków", mają oni bowiem od razu kod argumentu do wklepania w postaci szesnastkowej.

GETDIGIT (0379h)

– podprogram służy do wprowadzenia z klawiatury cyfry kodu szesnastkowego (0..9,A..F) z jednoczesnym wypisaniem jej na wyświetlaczu na pozycji podanej w rejestrze B przed wywołaniem procedury. Praktycznym zastosowaniem jest wprowadzanie przez użytkownika programu danych liczbowych w postaci szesnastkowej (lub jak się to okaże za chwilę także dziesiętnej) w sytuacji gdy zachodzi taka potrzeba, np. przy podawaniu adresu obszaru w zewn. pamięci RAM procesora.
– adres wywołania: 0379h
– we: B – numer pozycji na wyświetlaczu (1...8)
– wy: wprowadzona cyfra (0...15) 0...9, A...F
– traci: rejestr R0
– przykład: niech w naszym programie zajdzie potrzeba wprowadzenia liczby graczy, dzięki naszemu podprogramowi można to osiągnąć elegancko za pomocą kilku poniższych linii:

```
.......     poprzednie działania programu
.......
LCALL   CLS             ;wyczyszczenie displeja
MOV     DPTR, #napis_play   ;adres napisu „PLAY-"
MOV     B, #1           ;od 1-szej pozycji
LCALL   TEXT            ;wypisz psudo-tekst
MOV     B, #6           ;na 6-tej pozycji
LCALL   GETDIGIT        ;pobierz liczbę graczy
ANL     A, #7           ;i ogranicz ja w zakresie 0...7
JZkoniec ;jeżeli liczba graczy = 0 to zakończ program
MOV     B, A            ;jeżeli nie to umieść ją w rejestrze B
.........               ;....i baw się dalej
koniec:
.........   tu zakończenie programu
```

Nie zapomnijmy zdefiniować psudo-napisu „PLAY", który daje znać że program czeka na podanie ilości graczy, a więc:

```
napis_play      DB      _P, _L, _A, _Y, _minus, 0
```

W efekcie działania instrukcji z przykładu, na początku profilaktycznie wyczyszczony zostanie wyświetlacz, potem procesor wypisze komunikat „PLAY", informując użytkownika o potrzebie wprowadzenia liczby graczy, następnie po wciśnięciu klawisza, program ograniczy liczbę zawodników do 7-miu i sprawdzi ilu graczy wybrano. Jeżeli ich liczba jest = 0 to program zakończy się, jeżeli nie to rozpocznie się jego dalsza część.

GETACC (03A7h)

– podprogram pochodny od GETDIGIT. W wyniku jego wywołania komputerek oczekuje od użytkownika wprowadzenia 8-bitowej liczby szesnastkowej (2 cyfry) a następnie wprowadza ją do akumulatora.

Dodatkowo można określić pozycję na wyświetlaczu od której ma być wypisana wprowadzona liczba.
– adres wywołania: 03A7h
– we: B – pozycja displeja, od której ma być wyświetlana wprowadzana liczba (1...7)
– wy: Acc – wprowadzona liczba (np. 64h po wciśnięciu klawiszy '6' i '4')
– traci: rejestr R0
– przykład: program prosi o podanie 2 liczb 8-bitowych, następnie je mnoży, a wynik wypisuje na wyświetlaczu

```
LCALL   CLS             ;profilaktycznie czyścimy pole odczytowe
MOV     DPTR, #napis1   ;zaproszenie do wpisania 1 składnika
MOV     B, #1           ;od pozycji 1 displeja
LCALL   TEXT            ;wypisz komunikat
MOV     B, #4           ;od 4-tej pozycji displeja
LCALL   GETACC          ;pobierz 1-szy składnik
PUSH    Acc             ;i zapamiętaj go na stosie
MOV     DPTR, #napis2   ;zaproszenie do wpisania 2 składnika
MOV     B, #1           ;od pozycji 1 displeja
LCALL   TEXT            ;wypisz komunikat
MOV     B, #4           ;od 4-tej pozycji displeja
LCALL   GETACC          ;pobranie 2-ego składnika
LCALL   CLS             ;przygotowanie do wypisania wyniku
POP     B               ;odtwarzamy 1-szy składnik w rej. B
MUL     AB              ;pomnożenie składników (drugi jest w Acc)
MOV     DPH, B          ;przepisujemy wynik mnożenia B.A
MOV     DPL, A          ;do rejestru DPTR celem wypisania wyniku
MOV     B, #5           ;od 5-tej pozycji displeja
LCALL   DPTR4HEX        ;wypisanie wyniku mnożenia
.....
```

Nie zapomnijmy zdefiniować komunikatów zapraszających do wprowadzenia pierwszego, a następnie drugiego wskaźnika, a więc np.:

```
napis1  DB      _L, _1, _minus, 0   ;napis: „L1 -" (składnik nr 1)
napis2  DB      _L, _2, _minus, 0   ;napis: „L2 -" (składnik nr 2)
```

GETDPTR (03B9h)

– podprogram służący do wprowadzenia przez użytkownika 16-bitowej liczby (4 cyfry) z jednoczesnym wyświetleniem jej na displeju od pozycji określonej w rejestrze B. Wynik wprowadzania zostaje umieszczony w rejestrze DPTR. Procedura pochodna od GETACC.
– adres wywołania: 03B9h
– we: B – pozycja displeja, od której ma być wyświetlana wprowadzana liczba (1...5)
– wy: DPTR – wprowadzona liczba (np. AC81h po wciśnięciu klawiszy 'A', 'C', '8 i '1)
– traci: rejestr R0
– przykład: spróbujmy zapisać działanie programu podobne do funkcji „FILL" dostępnej z poziomu monitora komputerka edukacyjnego, oto instrukcje, w wyniku których program pyta o koniec i początek obszaru zewn. pamięci RAM do wypełnienia oraz o stałą wypełnienia, a następnie zapełnia wszystkie komórki podaną wartością. Wersja „surowa" bez bajeranckich komunikatów.

```
nasz_fill:
    LCALL   CLS             ;to już znamy
    MOV     B, #1           ;od pozycji 1 displeja
    LCALL   GETDPTR         ;pobierz najpierw koniec obszaru
    MOV     R7, DPH         ;i zapamiętaj go w rejestrach R7.R6
    MOV     R6, DPL
    LCALL   CLS             ;to już znamy
    MOV     B, #1           ;także od pozycji 1
    LCALL   GETDPTR         ;pobierz początek obszaru
    MOV     B, #7
    LCALL   GETACC          ;i pobierz stałą do wypełnienia
    MOV     B, A
wypelniaj:
    MOV     A, B
    MOVX    @DPTR, A
    INC     DPTR
    MOV     A, DPL
    CJNE    A, DPL, wypelniaj
    MOV     A, DPH
    CJNE    A, DPH, wypelniaj
koniec_wypelniania:
    .........
    .........
```

Przykład byłby krótszy, gdyby istniała instrukcja 8051 typu:

CJNE Rn, adres, ofset

zastanów się dlaczego i spróbuj uzupełnić program o niezbędne komunikaty.

W przykładzie dla uproszczenia algorytmu najpierw wprowadza się adres końcowy obszaru do wypełnienia, a następnie początkowy, to też temat na krótkie zastanowienie się. O ile linii program byłby dłuższy, gdyby było odwrotnie – najpierw początek a potem koniec obszaru pamięci?

I to już wszystkie podprogramy standardowe dotyczące podstawowych urządzeń wejścia-wyjścia naszego komputerka edukacyjnego, czyli klawiatury i wyświetlacza. Pozostało nam jeszcze jedno specjalne i jednocześnie dość użyteczne urządzenie wbudowane standardowo w procedur 8051 a pozwalające komunikować się naszemu komputerkowi ze światem zewnętrznym chodzi mianowicie o potocznie nazywany **port szeregowy**.

Zauważcie że na płytce waszego zestawu znajduje się kompletny układ do takiej transmisji (U13 wraz z C7...C10) oraz do połączenia komputerka wprost do portu szeregowego komputerka PC lub każdego innego wyposażonego w taki układ. Może to być także inny komputerek AVT2250 – to podpowiedź dla „ręczniaków".

W programie monitora znajdują się 3 dodatkowe podprogramy ułatwiające komunikowanie się naszego komputerka z innym urządzeniem port transmisji szeregowej, omówimy je za chwilę.

Chce w tym miejscu wyjaśnić że podane w tej części artykułu informacje na temat portu szeregowego nie wyczerpując tego tematu, mają jedynie za zadanie wyjaśnić działanie opisanych podprogramów. Dlatego bez wchodzenia w szczegóły (na które przyjdzie pora w jednym z kolejnych odcinków naszego cyklu) pokażę praktyczny sposób zmuszenia naszego komputerka do odbierania i wysyłania danych poprzez port szeregowy.

Czyż to nie wspaniałe móc dołączyć nasze malutkie „urządzonko" do zwykłego PC-ta, Amigi lub innego urządzenia zawierającego układ asynchronicznej transmisji szeregowej, tak drodzy Czytelnicy, za chwilę się to stanie.

Dla porządku powiem że przed rozpoczęciem zabawy z portem szeregowym i wywołaniem związanych z nim podprogramów monitora, powiem że będziemy korzystać z typu transmisji który można określić następującymi cechami:
– **asynchroniczna**: oznacza to że urządzenie nadawcze jak i odbiorcze musza mieć ustaloną tę samą szybkość transmisji, czyli że w takim samym tempie musza odbierać i nadawać kolejne bity danych. Niestety aby tak było najprościej jest ustawić ręcznie szybkość transmisji w sposób ręczny. W przypadku naszego komputerka najłatwiej jest tego dokonać za pomocą funkcji monitora „BAUD" – klawisz „B". W „przyrodzie" przyjęto pewne standardowe szybkości transmisji szeregowej asynchronicznej, a mianowicie:
  1200, 2400, 4800, 9600, 19200, 38400, 57600 bodów. Liczby te oznaczają ilość przesyłanych bitów w ciągu jednej sekundy, w żargonie komputerowców nazywa się je jednostką „bod" (ang. „baud"). Profesjonaliści na prezentacjach mówią o „bitach na sekundę" (B/sek) lub o kilobitach / sekundę.
  Zakres standardowych prędkości jest szerszy, tak w dół jak i górę, lecz my w naszej praktyce będziemy używać podanych wyżej, co w zupełności wystarczy każdemu „zjadaczowi chle...", chyba „zjadaczowi kompute...", sami zresztą dokończcie.
– w typowej transmisji zakładamy, że dana jest liczbą 8-bitową (można zatem przesyłać znaki z zakresu 0...255) dodatkowo występuje tzw. bit startu (logiczne „0") oraz bity stopu, u nas będzie to jeden (logiczne 1). Tak jak wspomniałem wcześniej o szczegółach transmisji dowiemy się później, toteż proszę się nie denerwować chwilowym brakiem wiedzy.

W praktyce wszystko co trzeba zrobić aby procesor 8051 zaczął odbierać i wysyłać znaki w tym trybie, należy wpisać odpowiednie wartości do kilku rejestrów specjalnych (SFR). Na szczęście program monitora komputerka edukacyjnego jest tak skonstruowany, że po włączeniu zasilania automatycznie ustawia wszystkie niezbędne rejestry tak jak trzeba, toteż pozostaje tylko skorzystać z dobrodziejstw transmisji szeregowej i zabrać się do roboty.

Do tego potrzebny będzie kabel, którego konstrukcję sposób wykonania przedstawiłem przy okazji prezentacji konstrukcji komputerka w numerach EdW z poprzedniego roku.

Tak a propos, to musze przyznać że choć wspomniany kabelek składa się tylko z dwóch identycznych wtyczek DB9 oraz 3 kabelków, to z prawidłowym jego wykonaniem macie często sporo problemów. Tak na prawdę to nie wiem dlaczego, toteż proszę o wiadomości, piszcie drodzy czytelnicy, piszcie... .

Zanim przejdę do omówienia wspomnianych podprogramów, muszę wyjaśnić że zmianę szybkości transmisji asynchronicznej naszego komputerka oprócz metody ręcznej, można osiągnąć także poprzez modyfikację rejestru specjalnego TH1 (SFR, adres: 8Dh). W zależności od żądanej prędkości transmisji należy wpisać do niego odpowiednią liczbę, zgodnie z wykazem poniżej:

prędkość transmisji	wartość TH1 (8Dh)
1200	D0h

```
2400        E8h
4800        F4h
9600        FAh
19200       FDh
38400       FEh
57600       FFh
```

W praktyce modyfikacji rejestru można dokonać za pomocą instrukcji:

```
MOV    TH1, #baud
```

gdzie za wyraz „baud" podstawiamy wybraną z tabeli powyżej wartość, np.

```
MOV    TH1, #FDh     ; ustawienie prędkości transmisji na
                       19200 bit/sek.
```

Z obsługą łącza szeregowego za pomocą wspomnianych procedur monitora, do omówienia których za chwilę przejdziemy, wiąże się dodatkowo rejestr przeterminowania transmisji, umieszczony w wewn. RAM procesora pod adresem 74h nazywany przeze mnie „overtime" (z ang. – „przeterminowanie").

Uwaga, rejestr ten nie jest standardowym rejestrem specjalnym SFR procesora 8051, został on zaimplementowany przeze mnie w trakcie tworzenia programu monitora dla komputerka AVT-2250, toteż jeżeli, ktoś np. postanowi sam zaprogramować sterowanie transmisją poprzez port szeregowy, powinien się z tym liczyć. Wnikliwi czytelnicy z pewnością a tym już wiedzą, ano choćby z informacji o adresie rejestru „overtime" równym 74h, a więc znajdującym się poza obszarem rejestrów specjalnych procesora SFR, mniej wtajemniczonym wszakże o tym przypominam.

W przypadku użycia procesor standardowych monitora czas przeterminowania określono na 60 sekund, co oznacza, że jeżeli w ciągu 1 minuty od wywołania podprogramu odebrania danej (bajtu) z portu szeregowego, dana ta nie zostania odebrana pomyślnie, to procedura zakończy się z ustawionym odpowiednim wskaźnikiem błędu. Dzięki temu użytkownik może stwierdzić fakt, że np. zewnętrzne urządzenie jest nieaktywne (np. wyłączone) nie może przesyłać danych, na skutek różnych szybkości transmisji odbiornika i nadajnika odbiór danej jest błędny.
Przejdźmy zatem do omówienia procedur obsługi łącza szeregowego.

INRS (02A5h)

- podprogram oczekuje na daną (bajt) z portu szeregowego, a po odebraniu umieszcza ją w akumulatorze (Acc). W przypadku gdy w ciągu minuty nie nadejdzie żaden znak, procedura kończy się automatycznie i dodatkowo zostaje ustawiony znacznik C, co świadczy o błędzie.
- adres wywołania: 02A5h
- we: bez parametrów
- wy: jeśli C=0 to odebrany znak znajduje się w Acc, w przeciwnym przypadku (C=1) wystąpił błąd – przeterminowanie
- zmienia: znacznik C
  (oraz dodatkowo informacja na przyszłość: zeruje znacznik RI oraz ustawia znacznik REN, znaczenie tych dwóch ostatnich poznamy przy okazji szczegółowego omawiania portu transmisji szeregowej procesora 8051).
- przykład: zaprogramujmy procesor tak aby odebrał 10 bajtów poprzez łącze szeregowe, a następnie zakończył program stwierdzając czy transmisja się powiodła, czy też nie.

```
b9600  EQU   0FAh              ;deklaracja liczby z tabeli szybkości

       MOV   TH1, #b9600       ;ustaw prędkość na 9600 bodów
       MOV   R7, #10           ;10 znaków do odebrania
       MOV   R0, #50h          ;bufor na 10 znaków od adresu 50h
następny:
       LCALL INRS              ;pobierz bajt z portu
       JC    blad              ;jeżeli wystąpił błąd to skocz do etykiety
       MOV   @R0, A            ;zapamiętaj daną w pamięci
       INC   R0                ;zwiększ adres wskaźnika bufora
       DJNZ  R7, nastepny      ;jeśli nie to następny bajt do odebrania
                                okej:
       MOV   DPTR, napis_OK ;komunikat o poprawnym odbiorze
pisz:  MOV   B, #1             ;od pierwszej pozycji
       LCALL TEXT              ;wypisz na displeju

stop:  SJMP  stop              ;stop programu

blad:  MOV   A, R0             ;pobierz wartość wskaźnika bufora
       ANL   A, #0Fh           ;zamień wartość wskaźnika R0 na liczbę
                                ;odebranych prawidłowo bajtów (0...9)
       MOV   B, #7             ;na pozycji 7-8
       LCALL A2HEX             ;pokaż liczbę odebranych znaków
       MOV   DPTR, #napis_ERR  ;komunikat o błędzie
       SJMP  pisz              ;skok do etykiety „pisz" = wypisanie
```

Należy jeszcze zdefiniować komunikaty o przebiegu transmisji, np.:

```
napis_OK DB     _S, _U, _C, _C, 0   ;"SUCC" skrót od „successfull" (ang. pomyślnie)
napis_ERR DB    _E, _r, _r, _o, _r, 0 ;to już znamy – patrz opis proc. TEXT
```

W przykładzie procesor odbiera 10 znaków (bajtów) z portu szeregowego ustawionego na szybkość 9600 bit/sek, i umieszcza je w wewn. pamięci RAM procesora w obszarze o adresach 50h....59h (patrz na wskaźnik R0). W przypadku prawidłowego odebrania wszystkich 10-ciu bajtów na wyświetlaczu wypisywany jest komunikat o pomyślnym przebiegu transmisji. Jeżeli zaś wystąpił błąd, na wyświetlaczu pojawia się komunikat o błędzie („Error") oraz dodatkowo zostaje wypisana liczba prawidłowo odebranych znaków. W kilku liniach zrealizowano uproszczoną konwersję liczby szesnastkowej na dziesiętną, korzystając z faktu, że w przypadku wystąpienia błędu nasz wskaźnik R0 będzie zawierał liczbę z przedziału 50...59h. Toteż poprzez logiczne wymnożenie (instrukcja „ANL") tej wartości przez liczbę 0Fh pozbywamy się niepotrzebnego starszego półbajtu „5", i zostaje nam tylko liczba z przedziału „0"..."9", czyli ilość prawidłowo odebranych znaków, którą następnie wypisujemy na displeju tuż za napisem „Error", prawda że eleganckie.

OUTRS (02B9h)

- podprogram natychmiast po wywołaniu ustawia port szeregowy w tryb nadawania (blokuje odbiornika), wysyła do portu bajt znajdujący się w akumulatorze (Acc), a na zakończenie ponownie odblokowuje odbiornik ustawiając w ten sposób port w tryb odbioru danych.
- adres wywołania: 02B9h
- we: A – znak do nadania
- wy: bez parametrów
- zmienia: (informacje zaawansowane) zeruje znacznik TI oraz po zakończeniu ustawia znacznik REN (odblokowuje odbiornik)
- przykład: poniżej zrealizujemy tzw. „efekt echa", dzięki któremu możliwe jest łatwe sprawdzanie obu linii (odbiorczej i nadawczej) portu szeregowego. Wszystkie odebrane znaki będą natychmiast wysyłane przez nasz komputerek z powrotem do urządzenia zewnętrznego, które powinno być ustawione w odpowiedni tryb pracy. Dodatkowo odebranie bajtu o kodzie 27 (komputerowcy w tym miejscu rozpoznają klawisz „Esc") kończy program.

```
nastepny:
       LCALL  INRS              ;czekanie na daną z portu
       JC     blad              ;jeżeli przeterminowanie to skocz
       LCALL  OUTRS             ;jeśli nie to odeslij znak
       CJNE   A, #27, nastepny  ;czy bajt = 27, nie to odbierz następny
       SJMP   stop              ;tak to skok na koniec
blad:  MOV    DPTR, #napis_ERR  ;komunikat o błędzie
       MOV    B, #1             ;na pozycji 1 displeja
       LCALL  TEXT              ;wypisz
stop:  SJMP   stop              ;i zakończ program

napis_ERR  DB                   _E, _r, _r, _o, _r, 0
```

INACCRS (02F2h)

- podprogram oczekuje na 2 znaki ASCII z portu szeregowego a następnie zamienia je na bajt i umieszcza wynik w akumulatorze. Odbierane znaki muszą reprezentować cyfry kodu szesnastkowego, czyli '0 ... 9, 'A ... F. Kody tych znaków – czyli ich reprezentacje liczbowe podałem wcześniej w artykule przy okazji omawiania procedury CONIN. Obowiązuje zasada z przeterminowaniem, tak jak w przypadku procedury INRS. Pierwszy odebrany kod jest traktowany jako starszy półbajt liczby, drugi 0– jako młodszy.
- adres wywołania: 02F2h
- we: bez parametrów
- wy: C=0 to w Acc znajduje się liczba (np. 8Eh po odbiorze liczb: 38h (znak '8) i 45h (znak 'E)), w przeciwnym przypadku (C=1) transmisja się nie powiodła.
- zmienia: uwagi jak w przypadku procedury INRS.
- przykład: ponieważ użycie tej procedury jest trywialne, pokażę w jaki sposób wykonać podprogram odwrotny, czyli taki który wysyła liczbę znajdująca się w akumulatorze poprzez port szeregowy jako 2 znaki ASCII, oto instrukcje:

```
OUTACCRS:                ;tak nazwiemy nasz przykład
   MOV    DPTR, #tabela  ;pobranie adresu wskaźnika tabeli
                          ;kodów ASCII
   PUSH   Acc            ;zapamiętanie liczby do wysłania
   SWAP   A              ;zamiana półbajtów
   ANL    A, #0Fh        ;i obliczenie ofsetu w tabeli
   MOVC   A, @A+DPTR     ;pobranie znaku z tabeli
   LCALL  OUTRS          ;i wysłanie – starszy półbajt
   POP    Acc            ;odtworzenie akumulatora
   ANL    A, #0Fh        ;obliczenie ofsetu w tabeli
   MOVC   A, @A+DPTR     ;pobranie znaku z tabeli
```

```
    LCALL     OUTRS          ;o wysłanie młodszego półbajtu
    .......                  ;dalsze instrukcje programu lub „RET"

tabela    DB      '0123456789ABCDEF
```

W przykładzie do konwersji liczby 8-bitowej (np. 49h) posłużono się dodatkową tabelą zdefiniowaną jako ciąg znaków ASCII odpowiadający kolejnym cyfrom kodu szesnastkowego, a więc: 0...9, A...F. Zauważmy że tabela po przetłumaczeniu na kod wynikowy będzie miała postać:

```
tabela    DB      30h,31h,32h,33h,34h,35h,36h,37h,38h,39h
          DB      41h, 42h,43h,44h,45h,46h
```

Tak więc np. w przypadku gdy wywołamy wspomniany przykład z liczbą w akumulatorze np. 4Ch, to najpierw program pobierze z tabeli bajt z przesunięciem +4 czyli 34h (co odpowiada znakowi '4), i wyśle go poprzez port szeregowy, a następnie to samo zrobi z młodszym półbajtem – 'C. W efekcie zewnętrzne urządzenie odbiorcze odbierze znaki: '4 i 'C. Proponuję przeanalizować dokładnie jeszcze raz cały przykład.

I to już wszystkie procedury monitora z których można korzystać i które pozwolą na skrócenie czasu pisania programu przez Ciebie, drogi Czytelniku. W naszej szkole spotkamy się z wieloma innymi przykładami, które przy okazji bardziej wyrafinowanych programów będziemy wspólnie omawiać, a następnie stosować.

Dodatkowe rejestry

Przy okazji omawiania zasobów systemowego „bios-a" naszego komputerka nie sposób nie wspomnieć o dodatkowych rejestrach (komórkach wewn. RAM) które wykorzystywane są przez monitor do jego pracy. Musisz wszakże, drogi Czytelniku, zdać sobie sprawę, że nawet wtedy gdy wydaje Ci się że komputerek nie robi nic, to jednak tak nie jest, w każdej sekundzie wykonywanych jest kilka tysięcy operacji, których zadaniem jest chociażby ciągłe kontrolowanie stanu wszystkich klawiszy oraz przemiatanie wszystkich pozycji wyświetlacza. Do tego wszystkiego potrzebne są niektóre komórki pamięci w obszarze wewnętrznej RAM procesora. Autor pisząc program monitora starał się ograniczyć do minimum ilość tych komórek, z których korzysta monitor, tak aby użytkownik komputerka miał do dyspozycji jak największy obszar z 128 bajtów wewnętrznej RAM procesora 8051. To tak jak z pamięcią operacyjną prawdziwych komputerów PC (informacja dla komputerowców), gdy jej brakuje (bo jest zajmowana przez mniej lub bardziej rozbudowane programu rezydentne – TSR), to inne uruchamiane programy mają mniej pamięci do dyspozycji, a często w ogóle nie mogą działać.

W przypadku naszego komputerka ze 128 bajtów wewn. RAM procesora monitor zajmuje 16 położonych „najwyżej" – tzn. adresy 70h...7Fh. Toteż nie należy nieświadomie w tym obszarze umieszczać swoich danych, modyfikując tym samym istotne dla działania całego systemu dane. Przypadkowa i nieprawidłowa modyfikacja jednej z tych komórek może nawet spowodować zawieszenie się komputerka i konieczność jego zresetowania przyciskiem „reset".

Dlatego warto poświęcić trochę czasu i zapoznać się z rejestrami wykorzystywanymi przez monitor. Powinieneś tez wiedzieć drogi Czytelniku że znajomość funkcji tych rejestrów ułatwi nam wspólna analizę przykładów publikowanych w naszym cyklu oraz ułatwi realizowanie wielu pożytecznych funkcji podczas pisania własnych programów.

DL1...DL8 – rejestry bufora wyświetlanych znaków

```
DL1     equ   78h    ;komorka znaku do wyswietlenia na DL1
DL2     equ   79h    ;j/w lecz na DL2
DL3     equ   7Ah    ;j/w lecz na DL3
DL4     equ   7Bh    ;j/w lecz na DL4
DL5     equ   7Ch    ;j/w lecz na DL5
DL6     equ   7Dh    ;j/w lecz na DL6
DL7     equ   7Eh    ;j/w lecz na DL7
DL8     equ   7Fh    ;j/w lecz na DL8
```

Powyżej podano definicje adresów komórek (rejestrów) w wewn. RAM procesora, które używane są do wyświetlania znaków na poszczególnych pozycjach wyświetlacza. I tak jeżeli wpiszemy jakąś liczbę (8-bit) do rejestru DL4, to na czwartym wyświetlaczu zapalone zostaną segmenty odpowiadające ustawionym pozycjom bitów w tej liczbie, zgodnie ze schematem opisanym w poprzednim odcinku naszego cyklu.

Modyfikując bezpośrednio te rejestry możemy umieszczać różne napisy lub pojedyncze znaki na wyświetlaczu. Oto kilka przykładów:

a) instrukcja

```
MOV   DL1, #255
```

spowoduje zapalenie wszystkich segmentów wyświetlacz DL1 (także kropki)

b) instrukcje

```
MOV   DL1, #0
MOV   DL2, #0
MOV   DL3, #0
MOV   DL4, #0
MOV   DL5, #0
MOV   DL6, #0
MOV   DL7, #0
MOV   DL8, #0
```

spowodują to samo co podprogram CLS – wyczyszczenie wyświetlacza.

c) wykonanie instrukcji

```
MOV   DL1, #_H
MOV   DL2, #_E
MOV   DL3, #_L
MOV   DL4, #_L
MOV   DL5, $_0
```

spowoduje pojawienie się znajomego napisu „HELLO" na wyświetlaczu.

W przypadku rejestrów DL1...DL8 ich przypadkowa modyfikacja nie spowoduje w żadnym przypadku zawieszenia systemu, jedynym efektem ubocznym może być wyświetlanie przypadkowych znaków i symboli.

CNT256 – rejestr licznika wyświetlacza (adres: 77h)

Rejestr ten jest automatycznie inkrementowany 256 razy na sekundę, czyli co 1 sekundę następuje jego wyzerowanie. Warto wiedzieć że trzy najmłodsze bity tego rejestru wykorzystywane są do określenia aktualnie aktywnej (w trybie multipleksowania) pozycji wyświetlacza (1 z 8-miu). Dlatego rejestru tego nie należy pod żadnym pozorem zapisywać instrukcjami typu:

```
MOV     CNT256, .....
```

może to bowiem zakłócić kolejność wyświetlania informacji na displeju, lub spowodować migotanie wyświetlaczy.

Rejestr ten można oczywiście odczytywać, w pewnych zastosowaniach może on posłużyć jako generator 8-bitowych liczb pseudolosowych, kiedy to np. liczba taka generowana jest po naciśnięciu klawisza przez użytkownika. Ze względu na dość częste zmiany zawartości tego rejestru (256 Hz) trudno jest przewidzieć potencjalnemu operatorowi, kiedy powinien wcisnąć klawisz aby uzyskać konkretną liczbę.

KLAWISZ – rejestr przechowujący kod wciśniętego klawisza (adres: 76h)

W tej komórce pamięci znajduje się kod ostatnio naciśniętego klawisza klawiatury komputerka. Istotne jest to że po zwolnieniu klawisza rejestr ten nie jest automatycznie zerowany. Można to zrobić samodzielnie instrukcją:

```
MOV     KLAWISZ, #0
```

po odczytaniu klawisza instrukcją : MOV A, KLAWISZ.

Korzystając z podprogramu CONIN nie jest to konieczne, bowiem wyzerowanie następuje każdorazowo po wywołaniu tej procedury.

CNTDEL – rejestr do generowania opóźnień przez procedurę DELAY (adres: 75h)

Rejestr ten jest używany przez procedurę DELAY do generowania opóźnień programowych, zgodnie z opisem przedstawionym wcześniej przy okazji omawiania tego podprogramu. Jeżeli do tego rejestru wpiszemy jakąś liczbę to będzie on automatycznie (z częstotliwością 256Hz) dekrementowany aż do wartości 0. Po osiągnięciu zera rejestr nie jest dalej modyfikowany. Rejestr ten może być modyfikowany dowolnie, jednak należy mieć świadomość że jest on automatycznie zmniejszany z podaną wcześniej częstotliwością.

OVERTIME – rejestr przeterminowania (adres: 74h)

Rejestr wykorzystywany do określania faktu przeterminowania jakiegoś procesu – np. odbioru znaku z łącza szeregowego. Może być wykorzystywany do innych celów. Należy jednak mieć świadomość że w przypadku wpisania jakiejś liczby (różnej od 0) rejestr ten jest automatycznie dekrementowany dokładnie co jedną sekundę. Aby jednak wykorzystać rejestr do odmierzania dłuższego odcinka czasu (z zakresu 1...255 sekund) należy przedtem zsynchronizować moment rozpoczęcia odliczania (wpisanie liczby do rejestru OVERTIME) z momentem zmniejszenia o 1, tak aby pierwsza dekrementacja nastąpiła dokładnie po 1 sekundzie od momentu wpisu. Najłatwiej tego dokonać testując zawartość rejestru CNT256. Jeżeli w momencie odczytu wynosi ona 00h, to możemy być pewni że pierwsze zmniejszenie nastąpi dokładnie po 1 sekundzie. A oto przykład praktyczny – odliczenie podanej liczby sekund (liczbę tę należy podstawić za nazwę: „liczba_sekund").

```
czekaj:
    MOV     A, CNT256     ;testowanie rejestru cnt256
    JNZ     czekaj        ;jeżli <>0 to testuj dalej
                          ;jeśli cnt256=0 to rozpocznij odliczanie
    MOV     OVERTIME,     #liczba_sekund; załadowanie liczby sekund
czekaj2:
    MOV     A, OVERTIME   ;testowanie rejestru „overtime"
    JNZ     czekaj2       ;jeżeli nie równy 0 to testuj dalej
```

..... ;tu dalsze instrukcje po odliczeniu sekund

OVERCONST – rejestr obsługi przeterminowania portu szeregowego (adres: 73h)

Komórka przechowująca czas przeterminowania (w sekundach) przy odbiorze znaku z portu szeregowego. Wartość domyślna to liczba 60, czyli 60 sekund. Wpisując inną wartość z zakresu 1...255 można ten czas modyfikować. Wartość z tej komórki jest automatycznie przepisywana do rejestru OVERTIME po wywołaniu podprogramu INRS – odbierającej znak z portu szeregowego

Przykład: wykonując instrukcję

MOV OVERCONST, #10

zmieniamy czas przeterminowania przy odbiorze z portu szeregowego z 60 na 10 sekund.

INTVEC – rejestr tablicy wektorów przerwań (adres: 72h)

Ze względu na fakt że nie zajmowaliśmy się do tej pory szczegółowym opisem układu przerwań procesora 8051 znaczenie tego rejestru okaże się szczególnie ważne gdy zapoznasz się z tą częścią układu. Dla porządku powiem tylko że komórka ta przechowuje wartość bardziej znaczącego bajtu, 16-bitowego adresu skoku do tablicy procedur obsługi przerwań zdefiniowanej przez użytkownika w przestrzeni adresowej procesora – zewnętrznej pamięci danych (układ SRAM – U4).

Wartość wpisywana tu powinna być z reguły równa początkowemu segmentowi pamięci U4, ustawianej przez zworę JP3 – patrz schemat elektryczny systemu AVT-2250.

W chwili obecnej, bez znajomości układu przerwań, szczegółowe wyjaśnianie znaczenia rejestru jest bezcelowe. Zgodnie z przedstawioną wcześniej zasadą, radzę bez znajomości i działania tej komórki pamięci, nie modyfikować jej nieświadomie.

BLINKS – rejestr atrybutów pozycji wyświetlacza (adres: 71h)

Znaczenie tego rejestru omówiliśmy przy okazji poprzedniej lekcji 5 szkoły mikroprocesorowej. Dla porządku przypomnę tylko że poszczególne bity tego rejestru odpowiadają za atrybut wyświetlanego na displeju znaku. Ustawienie np. najstarszego bitu w tym słowie powoduje że zapisany do rejestru DL8 znak będzie migotał. Częstotliwość migania określona jest wewnętrznie przez program monitora na 2Hz. Przyporządkowanie poszczególnych bitów pozycjom wyświetlacza jest następujące.

DL 1-2-3-4-5-6-7-8
bity: 7-6-5-4-3-2-1-0

Przykład zastosowania podałem podczas ostatniej lekcji 5 w poprzednim odcinku szkoły mikroprocesorowej.

A teraz ostatni rejestr wykorzystywany przez monitor. Dzięki niemu możliwe jest ingerowanie w sposób działania procedury obsługującej wyświetlacz i klawiaturę do tego stopnia, że można ją programowo odłączyć i wykorzystać do własnych celów. Mowa o procedurze przerwania generowanej przy przepełnieniu licznika T0.

BSW – rejestr specjalny monitora systemu AVT2250 (adres: 70h)

Komórka przechowująca słowo specjalne monitora systemu AVT-2250, tzw. „Bios Status Word". Każdy z bitów tego rejestru ma szczególne znaczenie, dlatego zapisując tę komórkę należy robić to bardzo ostrożnie. A oto nazewnictwo i znaczenie poszczególnych bitów.

7	6	5	4	3	2	1	0
int/ext T0	ext T0	-	-	1/4 Hz	1/2 Hz	1 Hz	2 Hz

bit 7: **int/extT0** – ustawienie bitu spowoduje skok pod adres xx0Bh, (gdzie xx to wartość rejestru INTVEC – adres 72h), po wykonaniu procedury obsługi wyświetlacza i klawiatury pod tym adresem powinna znajdować się dalsza cześć tej procedury lub instrukcja „RETI „ – powrotu z procedury obsługi przerwania

bit 6: **extT0** – ustawienie powoduje zaniechanie wykonywania procedury obsługi wyświetlacza i klawiatury zawartej w monitorze i skok pod adres w pamięci zewnętrznej równy xx00h, gdzie xx to wartość ze zmiennej INTVEC (adres 72h). Bit ten ma wyższy priorytet od bitu int/extT0

bity 5 i 4: **nie używane**

bit 3: **1/4 Hz** – w przypadku gdy bit „extT0" jest wyzerowany, bit ten zmienia swoją wartość co 4 sekundy (0,25 Hz)

bit 2: **1/2 Hz** – j/w lecz co 2 sekundy (0,5 Hz)

bit 1: **1Hz** – j/w lecz co 1 sekundę (1 Hz)

bit 0: **2Hz** – j/w lecz co 0,5 sekundy (2 Hz)

I tak np. z bitu BSW.0 korzysta funkcja migotania wyświetlaczy, uaktywniana za pomocą rejestru **BLINKS**.

Sposób korzystania szczególnie z dwóch najstarszych bitów rejestru BSW (int/extT0 oraz extT0) wyjaśniony zostanie dokładnie przy okazji omawiania układu przerwań procesora 8051. Bity 3,2,1 i 0 powinny być tylko odczytywane, co pozwala na realizację wielu ciekawych efektów związanych z dokładnym odmierzaniem równych interwałów czasowych. Przykładowe algorytmy korzystające z dobrodziejstw tych bitów podam przy okazji następnych spotkań w szkole mikroprocesorowej.

Sławomir Surowiński

Lekcja 6

W dzisiejszej lekcji proponuję przeanalizowanie ciekawej procedury– podprogramu dodatkowych który będzie nieraz wykorzystywany przez nas w kolejnych tworzonych wspólnie programach. Komputerowcy mogą te przykłady przepisać umieszczając go w dowolnie nazwanym przez siebie zbiorze typu „*.INC".

Zadanie 1

Napisać podprogram zamieniający 16-bitową liczbę na jej postać dziesiętną.

Np. niech w DPTR będzie liczba 8A12h, w wyniku wywołania tej procedury powinniśmy otrzymać wynik w postaci dziesiętnej, czyli: 35346. Należy przy tym zauważyć, że o ile ilość cyfr liczby szesnastkowej to cztery, o tyle przy zamianie liczb powyżej 270Fh wynik, czyli postać dziesiętna będzie miała aż 5 cyfr. Wynika z tego że wynik operacji zamiany będzie składał się z trzech bajtów, bo np. zamieniając liczbę 90ach na postać dziesiętną otrzymamy wynik 37036 dziesiętnie, toteż:

liczba HEX	liczba DEC
90 AC	03 70 36

Jak widać liczba przed zamianą zajmuje 2 bajty, po zamianie aż 3. Oczywiście przy konwersji liczb poniżej 10000 trzeci najstarszy bajt będzie nieznaczący, lecz nasza procedura powinna obsługiwać liczby z pełnego zakresu 16 bitów czyli 0...65535.

Zasada konwersji w naszym przykładzie polega na testowaniu każdego z 16-tu bitów liczby, jeżeli jest on ustawiony (=1) to do wyniku dodawana jest liczba będąca potęgą dwójki, której stopień odpowiada numerowi testowanego bitu, tzn. że dla bitu 0 będzie to 2^0 czyli 1, dla bitu 1 będzie 2^1 = 2,..... itd, aż do bitu 15, gdzie 2^15 = 32768. Jeżeli zaś testowany bit jest równy 0 to wynik pozostaje bez zmian. Dodatkowo podczas dodawania wykorzystano korekcję dziesiętną, dzięki czemu, wynik ma postać dziesiętną, a o to nam przecież chodziło. Jeżeli ktoś chce może na kartce papieru przeprowadzić konwersję tą metodą, jest to dość proste, należy tylko operować na małych wartościach.

A oto kilkanaście linii, dzięki którym przeprowadzana jest konwersja.

Na początku definicje dodatkowych rejestrów, źródła i wyniku działania procedury. Oznaczmy w wewn. pamięci RAM procesora dwie grupy rejestrów, źródła jako parę rejestrów Data: **hiData** i **loData**. Do nich wpisywana będzie liczba do przekształcenia. Określimy też miejsce wyniku, czyli trzy rejestry **wyn3**, **wyn2**, **wyn1** gdzie po zakończeniu działania procedury będzie znajdował się wynik.

```
        loData  equ     21h
        hiData  equ     20h     ;wej: hiData,loData = XXXX h

        wyn1    equ     24h     ;wyj: wyn3,wyn2,
                                ;wyn1 = (0)XXXXX d
        wyn2    equ     23h
        wyn3    equ     22h

;Procedura zamiany 16-bitowej liczby z rejestrów hiData.loData
;na postać dziesiętną.
;we: hiData – starszy bajt liczby do przekształcenia
      loData – młodszy bajt tej liczby
;wy: wyn3 – najstarszy bajt liczby po przekształceniu
      wyn2 – starszy bajt liczby po przekształceniu
      wyn1 – młodszy bajt liczby po przekształceniu

DEC16:                          ;a oto i sama procedura
        mov     wyn1, #0
        mov     wyn2, #0        ;na początku
        mov     wyn3, #0        ;wyzerowanie wyniku
        mov     R0, #1
        mov     R1, #16         ;licznik bitów liczby 16-bitowej
        mov     R2, loData
        mov     DPTR, #tabl     ;pobranie adresu tabeli składników

again:          cjne    R1, #8, loD     ;czy testujemy młodszy
                                        ;czy starszy bajt
        mov     R2, hiData
loD:    mov     A, R2
        anl     A, R0
        jz      fini

        mov     A, R1           ;obliczenie właściewj liczby
                                ;z tabeli
        dec     A               ;dla odpowiedniego bitu
        mov     B, #3           ;tabela ma po 3 bajty
        mul     AB
        mov     R3, A
        inc     A               ;poprawka na adres
        inc     A
        movc    A, @A+DPTR      ;pobranie pierwszego bajtu liczby

        add     A, wyn1         ;dodanie go do wyniku
        da      A               ;z korekcją dziesiętną
        mov     wyn1, A
        jnc     poC
        mov     A, wyn2         ;dodanie staszego bajtu
                                ;do wyniku
        add     A, #1
        da      A               ;z korekcją dziesiętną
        mov     wyn2, A
        jnc     poC
        mov     A, wyn3         ;to samo z nastarszym bajtem
                                ;wyniku
        add     A, #1
        mov     wyn3, A

poC:    mov     A, R3
        inc     A               ;poprawka na adres
        movc    A, @A+DPTR
        jz      fini

        add     A, wyn2
        da      A
        mov     wyn2, A
        jnc     poC2
        mov     A, wyn3
        add     A, #1
        mov     wyn3, A

poC2:           mov     A,R3
        movc    A, @A+DPTR
        jz      fini
        add     A, wyn3
        mov     wyn3, A

fini:   mov     A, R0
        rl      A
        mov     R0, A
        djnz    R1, again
        ret
```

Na koniec nalezy jeszcze odpowiednio zdefiniować tabelę potęg dwójki, oto ona.

```
tabl    db      03h,27h,68h             ; 2 do 15
        db      01h,63h,84h             ; 2 do 14
        db      00h,81h,92h             ; 2 do 13
        db      00h,40h,96h             ; 2 do 12
        db      00h,20h,48h             ; 2 do 11
        db      00h,10h,24h             ; 2 do 10
        db      00h,05h,12h             ; 2 do 9
        db      00h,02h,56h             ; 2 do 8
        db      00h,01h,28h             ; 2 do 7
        db      00h,00h,64h             ; 2 do 6
        db      00h,00h,32h             ; 2 do 5
        db      00h,00h,16h             ; 2 do 4
        db      00h,00h,08h             ; 2 do 3
        db      00h,00h,04h             ; 2 do 2
        db      00h,00h,02h             ; 2 do 1
        db      00h,00h,01h             ; 2 do 0
```

I gotowe, teraż można bez obawy zamieniać liczby, np. tak:

```
MOV             hiData, DPH     ;pobierz liczbę
                                ;z DPTR
MOV             loData, DPL
LCALL   DEC16           ;wywołaj podprogram
MOV             A, wyn3         ; a nastepnie wypisz
                                ;wynik na displeju
MOV             B, #3
LCALL   A2HEX           ;najstarszy bajt
MOV             A, wyn2
MOV             B, #5
LCALL   A2HEX           ;starszy bajt
MOV             A, wyn1
MOV             B, #7
LCALL   A2HEX           ;i wreszcie młodszy bajt wyniku
```

W przykładzie zawartość rejestrów DPTR została zamieniana do postaci dziesiętnej a następnie wypisana na wyświetlaczu na pozycjach DL3...DL8.

Zadanie 2

I na deser małe zadanko, proszę samodzielnie spróbować napisać następujące procedury:

a) dodawania dwóch liczb 16-bitowych
b odejmowania dwóch liczb 16-bitowych
c) mnożenia liczby 16-bitowej przez 8-bitową (wynik będzie 24 bitowy)
d) dekrementacji z korekcją akumulatora

Za miesiąc rozwiązanie zadania, tymczasem życzę wesołej zabawy.

Sławomir Surowiński