Mikrokontrolery? To takie proste...

Część 10
Asembler – język maszynowy procesora

W niniejszym odcinku szkoły mikroprocesorowej kończymy omawianie listy instrukcji procesora 8051. Pozostało kilka prostych instrukcji używanych głównie w pętlach programowych oraz instrukcja pusta. W dalszej części artykułu zajmiemy się dokładnym opisem wszystkich podprogramów usługowych monitora – czyli programu komputerka edukacyjnego, który każdy z Was ma zapisany w EPROM-ie swego zestawu. Poznamy tez proste sposoby na wykorzystanie podstawowych zasobów tego programu, dzięki którym możliwa będzie kontrolowana ingerencja we wszystkie mechanizmy dostępne w Twoim systemie edukacyjnym.

Kontynuujemy opis instrukcji z grupy skoków. Pozostała nam do omówienia dość rzadko wykorzystywana przez początkujących programistów, instrukcja skoku względem wskaźnika danych DPTR, dwie instrukcje skoków warunkowych testujących zawartość akumulatora oraz dwa typy instrukcji, które dzięki swojej konstrukcji znajdują najszersze zastosowanie przy porównywaniu zawartości niektórych rejestrów z wartościami stałymi (wykorzystywane np. przy odczycie kodów naciśniętego klawisza) lub w pętlach programowych. Oto one:

JMP A+DPTR

– ang. “Jump Indirect relative to DPTR” – skocz pośrednio względem rejestru DPTR
– w wyniku tej instrukcji następuje skok pod adres będący sumą aktualnej wartości rejestru DPTR (liczba 16–bitowa) i wartości akumulatora (liczba 8–bitowa). Można powiedzieć że skok następuje pod adres w pamięci programu umieszczony w DPTR z przesunięciem podanym w akumulatorze. Przesuniecie to traktowane jest jako liczba bez znaku, czyli z zakresu <0...255>
  PC <– A + DPTR
– kod: 0 1 1 1 0 0 1 1 73h
– cykle: 2 bajty: 1
– przykład: realizacja skoków w miejsca w programie określone poprzez numer w zmiennej “pozycja” (umieszczonej – zadeklarowanej z wewn. RAM procesora)

```
    MOV  A, pozycja             ;załadowanie numeru pozycji
    MOV  B, #2                  ;2 bo instrukcje w tabeli skoków sa
                                2–bajtowe (AJMP)
    MUL  A, B                   ;obliczenie faktycznego ofsetu
                                do tabeli skoków
    MOV  DPTR, #tablica_skokow  ;załadowanie adresu
                                tabeli skoków do DPTR
    JMP  A+DPTR                 ;wykonanie skoku
tablica_skoków:                 ;tu zaczyna sie tabela skoków
    AJMP etyk01                 ;tu nastąpi skok gdy “pozycja” = 0
    AJMP etyk02                 ;tu nastąpi skok gdy “pozycja” = 1
    AJMP etyk03                 ;tu nastąpi skok gdy “pozycja” = 2
    AJMP etyk04                 ;tu nastąpi skok gdy “pozycja” = 3
    .....                       ;pozostałe instrukcje programu
    .....
etyk01:                         ;właściwe miejsce skoku według pozycji = 0
    .....                       ;tu mozna umieścić instrukcje
etyk02:                         ;właściwe miejsce skoku według pozycji = 1
    .....
etyk03:                         ;właściwe miejsce skoku według pozycji = 2
    .....
etyk04:                         ;właściwe miejsce skoku według pozycji = 3
    .....
```

Poniżej zapoznamy się z instrukcjami skoków warunkowych, które badają warunek zgodności zadanego bajtu w wew. RAM procesora (rejestru) z innym rejestrem lub argumentem bezpośrednim (liczbą). Dwie z nich JZ i JNZ sprawdzają czy w akumulatorze znajduje się liczba zero czy nie i na tej podstawie podejmowana jest decyzja o skoku. Pozostałe instrukcje porównujące wybrane rejestry z innym lub konkretną liczbą znajdują zastosowanie szczególnie w pętlach programowych, gdzie konieczne jest wykonanie n–razy określonej opracji.

JZ rel

– ang. “Jump if Accumulator is Zero”, skocz jeżeli w akumulatorze jest liczba 0 (zero)
– sprawdzana jest zawartość akumulatora (A), jeżeli jest równa zero, to do licznika rozkazów PC dodawane jest przesunięcie “rel” – na zasadach takich jak opisano wczesniej przy okazji omawiania poprzednich instrukcji skoków z argumentem “rel” (liczba 8–bitowa ze znakiem w kodzie U2).
  PC <– PC + 2, jeśli A = 0, to PC <– PC + rel
– kod: 0 1 1 0 0 0 0 0 60h
– cykle: 2 bajty: 2 (kod instrukcji 60h + przesunięcie “rel”)
– przykład:

```
    MOV  A, B        ;załadowanie rej. B do Acc celem
                     ;sprawdzenia czy = 0
    JZ   jest_zero   ;jeżeli tak to skok do etykiety “
                     jest_zero”
nie_zero:            ;jeżeli nie to wykonuj pozostałe instrukcje
    .....
    .....
jest_zero:           ;tu nastąpi skok jeżeli rej.B był równy zero
    .....            ;i zostaną wykonane te instrukcje
    .....
```

JNZ rel

– ang. “Jump if Accumulator is Not Zero”, skocz jeżeli akumulator nie jest = 0 (zero)
– sprawdzana jest zawartość akumulatora (A), jeżeli jest różna od zera, to do licznika rozkazów PC dodawane jest przesunięcie “rel” – na zasadach takich jak opisano wczesniej przy okazji omawiania poprzednich instrukcji skoków z argumentem “rel” (liczba 8–bitowa ze znakiem w kodzie U2).
PC <– PC + 2, jeśli A <> 0, to PC <– PC + rel
– kod: 0 1 1 1 0 0 0 0 70h
– cykle: 2 bajty: 2 (kod instrukcji 70h + przesuenięcie “rel”)
– przykład:

```
   MOV   A, B          ;załadowanie rej. B do Acc celem
                        sprawdzenia czy = 0
   JNZ   nie_zero      ;jeżeli nie to skok do etykiety “nie_zero”
jest_zero:             ;jeżeli tak to wykonuj pozostałe instrukcje
   .....
   .....
nie_zero:              ;tu nastapi skok jeżeli rej.B był różny od
                        zera
   .....               ;i zostaną wykonane te instrukcje
   .....
```

A oto wspomniana wczesniej grupa instrukcji używana głównie w pętlach programowych lub przy zwielokrotnionym sprawdzaniu warunków zgodnosci określonych rejestrów z innym lub z argumentami stałymi. Ogólna postać instrukcji jest następująca:

CJNE <arg1>, <arg2>, rel

– ang. “Compare and Jump if Not Equal”, porównaj i skocz jeżeli argumenty porównania nie są sobie równe

W instrukcji tej porównywane sa dwa argumenty: arg1 i arg2. Jeżeli nie są one równe (ich wartości nie sa równe), to do zawartości licznika rozkazów jest dodawane przesunięcie “rel” na zasadach zgodnych z omówionymi wcześniej. W efekcie zostaje wykonany skok w programie. Tu uwaga, skok nastepuje względem instrukcji występującej po instrukcji CJNE. Dodatkowo jest zmieniany znacznik przeniesienia C. I tak, jeżeli w wyniku porównania okaże sie zę argumrent arg1 jest mniejszy od argumentu arg2 to do znacznika C wpisywana jest jedynka (znacznik jest ustawiany), w przeciwnym przypadku znacznik jest zerowany (C=0). W zaleznosci od typu argumentów instrukcji możliwe są cztery przypadki, oto one.

CJNE A, adres, rel

– porównywany jest akumulator oraz komórka wew. RAM o adresie podanym bezposrednio jako argument “adres”
PC <– PC + 3, jesli A <> (adres), to PC <– PC + rel
dodatkowo: C <– 0 gdy A>= (adres), lub C<–1 gdy A< (adres)
– kod: 1 0 1 1 0 1 0 1 B5h
– cykle: 2 bajty: 3 (kod instrukcji + adres + przesunięcie “rel”)
– przykład:

```
   MOV   B,#56
check:
   CJNE  A, B, zwieksz
   SJMP  koniec
zwieksz:
   INC   A
   SJMP  check
koniec:
   CLR   B
   ......
```

W przykładzie tym do rejestru B wpisywana jest liczba 56. Nastepnie sprawdzany jest warunek zgodności akumulatora z rejestrem B, jeżeli nie są sobie równe, to akumulator jest inkrementowany (dodawana jest do niego jedynka) – patrz etykieta “zwieksz”. Następnie operacja jest powtarzana od początku – instrukcja “SJMP check”. Jeżeli w końcu nastąpi zgodność obu rejestrów, wykonywany jest skok do etykiety “koniec”, gdzie rejestr B zostaje wyzerowany.

CJNE A, #dana, rel

– akumulator zostaje porównany z argumentem bezpośrednim (8–bitową liczbą), jeżeli nie są zgodne nastepuje skok.
PC <– PC + 3, jeśli A <> dana, to PC <– PC + rel
dodatkowo: C <– 0 gdy A>= dana, lub C<–1 gdy A< dana
– kod: 1 0 1 1 0 1 0 0 B4h
– cykle: 2 bajty: 3 (kod instrukcji + dana + przesunięcie “rel”)
– przykład: niech w zmiennej (rejestrze) “klawisz” będzie przechowywany kod wciśnętego klawisza w systemie mikroprocesorowym (ot choćby w naszym komputerku). Jeżeli chcemy podjąć określone działanie w zależności od rodzaju klawisza, należy przechowywany kod klawisza porównać z konkretna liczbą.

```
czekaj:
   MOV   A, klawisz      ;pobranie kodu wciśniętego klawisza
   CJNE  A, #65, sprB    ;czy wcisnięto klawisz “A” (65 – kod “A”)
   ........              ;tak to wykonuj te instrukcje
sprB:
   CJNE  A, #66, sprC    ;nie to czy wciśnięto klawisz “B”
   ........              ;tak to wykonuj te instrukcje
sprC:
   CJNE  A,#67, sprD     ;nie to czy wciśnięto klawisz “C”
   .........             ;tak to wykonuj te instrukcje
sprD:
   CJNE  A, #68,czekaj   ;nie to czekaj na kolejne naciśnięcie
                          klawisza
   .........             ;tak to wykonuj te instrukcje
```

CJNE Rn, #dana, rel

– rejestr Rn (R0...R7) zostaje porównany z argumentem bezpośrednim, jeżeli nie sa zgodne zostaje wykonany skok
PC <– PC + 3, jeśli Rn <> dana, to PC <– PC + rel gdzie n = 0...7
dodatkowo: C <– 0 gdy Rn>= dana, lub C<–1 gdy Rn< dana
– kod: 1 0 1 1 0 n2 n1 n0 gdzie n2 n1 n0 określają jeden z rejestrów R0...R7 stąd kody: B8h....BFh
– cykle: 2 bajty: 3 (kod instrukcji + dana + przesuniecie “rel”)
– przykład:

```
   MOV   R4, P1            ;odczytanie stanów z portu P1
   CJNE  R4, #100, nie_100 ;porównanie ich z liczbą 100
   SETB  P3.0              ;równe to ustaw pin 0 portu P3
   ......
nie_100:
   CLR   P3.0              ;nie równe to zeruj pin 0 portu P3
   ......
   ......
```

W przykładzie porównywana jest zawartość rejestru R4 z liczbą 100, jeżeli występuje zgodność, to w porcie P3 zostaje ustawiony najmłodszy bit (pin) P3.0,w przeciwnym przypadku jest zerowany. Jest to prosty przykład komparatora liczby 8–bitowej podawanej na port P1 z zewnątrz ze stałą liczbą (w tym przypadku jest to liczba 100).

CJNE @Ri, #dana, rel

– porównywana jest zawartośc komórki w wew. RAM której adres znajduje sie w rejestrze Ri (R0 gdy i=0, lub R1 gdy i=1) z argumentem bezpośrednim. Jeżeli się różnią to następuje skok.
PC <– PC + 3, jeśli (Ri) <> dana, to PC <– PC + rel gdzie i = 0, 1
dodatkowo: C <– 0 gdy (Ri) >= dana, lub C<–1 gdy (Ri) < dana
– kod: 1 0 1 1 0 1 1 i gdzie i=0, 1 stąd kody: B6h, B7h
– cykle: 2 bajty: 3 (kod instrukcji + dana + przesuniecie “rel”)
– przykład: sekwencja instrukcji porównania w postaci:

```
   MOV   R1, #30h
   CJNE  @R1, #255, skocz
   .......
skocz:
   .......
```

jest równoważna sekwencji

```
   MOV   A, #255
   CJNE  A, 30h, skocz
   .......
skocz:
   .......
```

przeanalizuj i zastanów sie dlaczego. Podpowiem tylko, że w przykładzie porównywana jest zawartość komórki pamięci wew. RAM z określoną liczbą, przy różnicy wystepuje skok.

Ostatnią instrukcją skoków warunkowych jest polecnie DJNZ. Ogólna postać instrukcji jest następująca:

DJNZ <arg>, rel

– ang. “Decrement and Jump if Not Zero”, zmniejsz o jeden i skocz jeżeli nie równe zero

W wyniku tej operacji od wskazanego argumentu “arg” jest odejmowana jedynka (jest on dekrementowany). Jeżeli w wyniku odjęcia wartość argumentu nie jest równa zero, to zostaje wykonany skok zgodnie z zasadami opisanymi wcześniej w przypadku argumentu “rel”. Stan

znaczników nie zmienia się. W zaleznosci od typu argumentu rozróżnia się dwa typy instrukcji, oto one.

DJNZ Rn, rel

– zmniejszona zostaje zawartość podanego rejestru Rn (R0...R7) o jeden, a nastepnie jeżeli nie jest równa zero, to następuje skok.
  PC <– PC + 2, Rn <– Rn – 1, gdzie n = 0...7
  jeżeli Rn <> 0, to PC <– PC + rel
– kod: 1 1 0 1 1 n2 n1 n0 gdzie n2 n1 n0 określają jeden z rejestrów R0...R7 stąd kody: D8h...DFh
– cykle: 2 bajty: 2 (kod instrukcji + przesunięcie "rel")
– przykład: sekwencja instrukcji

```
   CLR    A
   MOV    R7, CLR A #26
dodaj:
   ADD    A, #1
   DJNZ   R7, dodaj
   ......
```

jest równoważna poleceniu

```
   ADD    A,#26
```

co w obu przypadkach powoduje dodanie do akumulatora liczby 26.

DJNZ adres, rel

– zmniejszona zostaje zawartość komórki pamięcie wew. RAM o podanym bezpośrednio adresie o jeden, a nastepnie jeżeli nie jest równa zero, to następuje skok.
  PC <– PC + 2, (adres) <– (adres) – 1, jeżeli (adres) <> 0, to PC <– PC + rel
– kod: 1 1 0 1 0 1 0 1 D5h
– cykle: 2 bajty: 3 (kod instrukcji + adres + przesunięcie "rel")
– przykład: sekwencja instrukcji przedstawiona poniżej daje taki sam efekt jak w porzednim przykładzie.

```
   CLR    A
   MOV    B, #26
dodaj:
   ADD    A, #1
   DJNZ   B, dodaj
   .......
```

A teraz pora na naprawdę ostatnią instrukcję z listy poleceń procesora 8051.

NOP

– ang. "No Operation", nie rób nic
– jest to instrukcja, w wyniku której nie zmienia się stan procesora, z wyjątkiem licznika rozkazów którym po pobraniu tej instrukcji jest zwiększany o jeden.
– kod: 0 0 0 0 0 0 0 0 00h
– cykle: 1 bajty: 1
– przykład: sekwencja instrukcji

```
   MOV    A, B
   NOP
   MUL    A, B
```

zajmie procesorowi o 1 cykl maszynowy więcej niz sekwencja

```
          MOV     A, B
          MUL     A, B
```

ale efekt działania będzie taki sam w obu przypadkach.

I na tym kończy się lista instrukcji procesora 8051. Teraz pozostaje juz tylko praktyczne ich wykorzystanie w celu tworzenia programów. Na początku będą to aplikacje na komputerek edukacyjny, który każdy z Was powinien już mieć uruchomiony, potem na dowolne układu elektroniczne, które już sami będziecie konstruować bazując na procesorach serii 8051 I pochodnych.

Pozostaje jeszcze do wyjaśnienia grupa poleceń nie bedących instrukcjami asemblera procesora 8051, lecz będąca deklaracjami przypisania i definiowania ciagów bajtów, akceptowanymi przez wiekszość kompilatorów (w tym zamieszczony na dyskietce AVT–2250/D program PASM51.EXE). Dzięki nim czytanie kodu programu przez programistę jest łatwiejsze, toteż powinni szczególnie pamiętać o tym "komputerowcy". Ze względu jednak na używanie takich poleceń w naszych przykładach, szczególną uwagę powinni zwrócić na te część artykułu także "ręczniacy".

Pierwszą uzyteczną deklaracją jest polecenie "**EQU**" (ang. "equal" – równy, taki sam, tożsamy) – przypisania nazwie występujacej po lewej stronie polecenia wartości z prawej strony, np.

```
   DL1           EQU            78h
```

oznacza że w dalszej części programu przez skrót "DL1" należy rozumieć liczbę szesnastkową 78h (120 dziesiętnie). I tak kompilator oraz "ręczniacy" powinny tłumaczyć (rozumieć) te 3 litery "DL1" jako liczbę 120. I tak jeżeli w dalszej części programu, juz w samym kodzie pojawi sie np. instrukcja:

```
   MOV               DL1, A
```

oznaczać to bedzie to samo co instrukcja

```
   MOV               78h, A
```

Podobnie np. jeżeli zechcemy np. załadować liczbę (nie zawartość komórki spod podanego adresu) 78h np. do rejestru B wykonamy operację:

```
   MOV               B, #DL1
```

co będzie tożsame jako

```
   MOV               B, #78h
```

Znaczek "#" oznacza że mamy do czynienia z argumentem bezpośrednim, czyli liczbą, a nie jak w przykładzie poprzednim z adresem komórki w wewnętrznej pamięci RAM procesora. Warto to zapamiętać.

Dla zapamiętania jeszcze jeden przykład: niech pod adresem F005h w obszarze zewnętrznej przestrzeni adresowej procesora (w jakimś układzie skonstruowanym przez nieznanego elektronika a wykorzystującym procesor 8051) znajduje się 8–bitowy rejestr pracujacy jako wyjście informacji, np. sterujący ośmioma przekaźnikami załączającymi żarówki węża świetlnego – chociażby znany wam już układ 74573. Aby zapalić co drugą żarówkę należy wykonać proste instrukcje zapisu do zewnętrznej pamięci danych, zakładając że rejestr jest zatrzaskiwany pod wpływem (nie bezpośrednio !!!) sygnału /WR – zapisu do ext. RAM procesora. Zadeklarujmy zatem ten specjalny adres jako nazwę pochodzaca od angielskiego słowa "wąż":

```
   SNAKE EQU           0F005h
```

Wykonanie w dalszej części programu sekwencji:

```
   MOV      DPTR, #SNAKE    ;załaduj adres do wskaźnika
                            danych
   MOV      A, #55h         ;w liczbie 55h sąsiednie bity
                            sa różne: 01010101
   MOVX     @DPTR, A        ;i zapisz do rejestru pod
                            wskazany adres
```

spowoduje zamierzony efekt. Pierwszą linię można zapisać oczywiście jako:

```
   MOV      DPTR, #F005h
   ......
   ......
```

ale jak sami widzicie pierwszy sposób jest bardziej czytelny, bowiem od razu wiemy czytając program, że w tym miejscu nastapi zapis do rejestru węża świetlnego.

Drugą deklaracją ważną szczególnie a może przede wszyskim dla komputerowców jest dyrektywa "**INCLUDE**" – ang. włącz w sensie dołącz, dopnij, weź pod uwagę kompilując program". Służy ona do włączania podczas kompilacji zbiorów dodatkowych), oprócz tego, którym jest celem kompilacji, przy wywołaniu programi kompilatora PASM51.EXE, w których znajdują się dodatkowe kawałki "kodu" źródłowego użytkownika. W sensie dosłownym mogą to być zbiory tekstowe zawierające biblioteki (w postaci źródłowej) dodatkowych procedur np. matematycznych.

W praktyce np. kiedy zachodzi potrzeba użycia podprogramu dodawania dwóch liczb 32–bitowych w pięciu aplikacjach (programach), nie warto umieszczać kodu tej procedury w każdym z pięciu zbiorów źródłowych. Lepiej i wygodniej jest raz zapisać kod źródłowy takiej procedury w odzielnym zbiorze np. pod nazwą: "*ADD32.INC*", po czym w każdym z pieciu aplikacji zapisać 1–liniową deklarację dołączenia tego zbioru podczas kompilacji w postaci:

```
   INCLUDE          ADD32.INC
```

co spowoduje dołączenie treści zbioru ADD32.INC do pliku głównego w miejscu jej wywołania. Oczywiście jeżeli w zbiorze z podprogramem wystąpi błąd w zapisie jakiejś instrukcji kompilator automatycznie przerwie tłumaczenie sygnalizując komunikatem odpowiedni błąd z podaniem nazwy zbioru włączonego dyrektywą "INCLUDE".

Jak wiecie, kod programu (już po przetłumaczeniu) na procesor składa się z ciągu bajtów instrukcji i danych. Czasami zachodzi potrzeba definiowania w kodzie programu (podczas pisania) tablic wartości stałych, np. opisujących przebieg sinusa dla kąta pełnego, lub numer ostatniego dnia każdego miesiąca. Program w czasie pracy za pomoca instrukcji "*MOVC A, @A+DPTR*" (patrz opis instrukcji) może pobrać takie dane i wykorzystać je do dalszych obliczeń. Jak zatem zdefiniowac takie ciągi bajtów w naszym programie, ano za pomoca dyrektyw: **DB** – definiujacej bajty, **DW** – definiujacej 16–bitowe słowa (podwójne bajty) oraz **DD** – definiującej 32–bitowe słowa (poczwórne bajty).

Przy tworzeniu takiego strumienia danych (stałych) wszystkie składniki, liczby musza być oddzielone przecinkami. Liczby mozna zapisywać

z ogólnie przyjętą zasadą (jak w całym asemblerze) w czterech różnych postaciach:
– dziesietnej, np. 23, 199, 45, 255, 54675
– szesnastkowej (na końcu litera "h"): 12h, 7Fh, 0ABh, 1234h, itp
– binarnej (na końcu litera "b"): 10101010b, 010b, 010010010010b
– za pomocą kodu znaku (znak w apostrofach): '4 – oznacza kod znaku ASCII "4" czyli fizycznie liczbę: 52. Sposób ten w przypadku kompilatora PASM51.EXE daje sie wykorzystać z dyrektywą "DB". Argumentami "DW" i "DD" mogą być tylko liczby zapisane w trzech poprzednich formatach.

Korzystając z ostatniego sposobu mozna zapisywać całe teksty (np. do wyświetlenia potem na dołączonym do komputerka – wyświetlaczu tekstowym LCD), lub po prostu informacje autorskie o danym programie, jego wersji, czy dacie powstania.

I tak np. jeżeli chcemy zdefiniowac tablicę cyfr wykorzystywanych w kodzie szesnastkowym, należy użyć dyrektywy DB w postaci np.

```
hextab DB        '0123456789ABCDEF'
```

co po przetłumaczeniu na kod maszynowy procesor zrozumie jako ciąg bajtów:

48 49 50 51 52 53 54 55 56 57 65 66 67 68 69 70

zapisanych dziesiętnie.

Czyli można zapisac ten ciąg w inny sposób np.:

```
hextab DB 48,49,50,51,52,53,54,55,56,57,65,66,67,68,69,70
```

W przypadku dyrektywy DW mozna w roli składników umieszczać liczby 16–bitowe czyli z zakresu: 0...65535, a w przypadku dyrektywy DD, liczby z zakresu 0... 4294967295, przykłady deklaracji mieszanych:

```
bigtab   DW   1234h, 65535, 1010101001010101b, 1, 0
longtab  DD   12345678h, 10000234,0101010b, 10000000000000000000001b
```

Zauważcie, że przed każdą deklaracją umieściłem jakiś wyraz, który jest po prostu etykietą opisującą deklarowane stałe. W programie jest to bardzo często niezbędne a nader wygodne. No bo jaki adres zapiszecie do wskaźnika DPTR przy uzyciu instrukcji "*MOVC A, @A+DPTR*", ręczniacy beda musieli po prostu policzyć wszystkie bajty znajdujące sie przed dana deklaracją tablicy i wpisać je po kodzie instrukcji MOV DPTR,# jakaś_liczba_16_bitowa, komputerowcy zastosuja polecenie, np.

```
MOV              DPTR, #bigtab
```

a kompilator PASM51 sam obliczy adres tablicy, gdziekolwiek ona by się znalazła, a następnie wstawi go za kodem instrukcji *MOV DPTR, #nn.*

W przypadku dyrektywy "DB" możliwe jest łączenie w jednej linii kilku rodzajów zapisów liczb, np. mieszając teksty z zapisem dziesiętnym, np.

```
tekst1           DB        16, 'WITAJ CZYTELNIKU'
```

A propos ten przykład może posłużyć jako ilustracja argumentu wywołania jakiejś procedury (podprogramu) w wyniku którego zostaje wypisany wskazany tekst. Pierwszy bajt w deklaracji tablicy określa fizyczną liczbę znaków w tablicy tekstowej – czyli długość napisu, dalej od razu wystepuje dany tekst, prawda że logiczne podejście. W ten prosty sposób procedura wie kiedy dany tekst się kończy. Istnieją także inne sposobu realizacji tego pomysłu, ale to nie jest w tej chwili tematem naszego artykułu.

Każdy program pisany i kompilowany przez komputerowców za pomocą programu PASM51.EXE musi się kończyć deklaracją "END", która mówi programowi, że w tym miejscu kończy się tekst źródłowy i resztę linii jeżeli one istnieją, należy zignorować.

Dodatkowo przy użyciu tego kompilatora w każdym programie źródłowym (głównym a nie w zbiorach z podprogramami typu INC) w pierwszej linii powinna znaleźć się dyrektywa deklarujaca procesor na który pisany jest dany program. Dyrektywa ta to : "CPU", za którą powinien znaleźć sie argument w postaci nazwy zbioru zawierającego deklaracje typu EQU – opisujące nazwy wszystkich rejestrów specjalnych oraz bitów i w ogóle całego nazewnictwa zwiazanego z wybraną kostką.

Na dyskietce AVT–2250/D znajduje sie taki zbiór definicyjny o nazwie *8052.DEF* warto go sobie obejrzeć, aby stwierdzić zgodność informacji przedstawionych przeze mnie w niniejszym artykule. Zbiór taki mozna oczywiście modyfikować, lecz na wstepnym etapie nauki programowania radze tego nie robić, a przynajmniej zrobić jego wierną kopię przed takimi eksperymentami.

Jak każdy program – tekst źródłowy powinien zawierać komentarze programisty, dzięki czemu późniejsza analiza programu jest łatwiejsza czy w ogóle możliwa do zrealizowania. Wiekszość asemblerów na procesory 8051 (ale nie tylko) rozpoznaje znak średnika ";" jako znak zaczynający w danej linii tekst komentarza. Toteż kompilator analizując linię po linii kod źródłowy programu i tłumaczac go, napotkawszy średnik ignoruje cały tekst od tego znaku aż do końca linii. Jak zdążyliście sie zorientować, komentarze mogą być umieszczane na początku linii wtedy cała linia jest komentarzem, lub po każdej linii z instrukcją dla procesora, co znacznie ułatwia późniejszą analize programu i wyszukiwanie błędów, a o modyfikacjach nie wspomnę. Przykład komentarza:

```
MOV   A, B        ;załadowanie zawartości rejestru B do akumulatora
```

Tekstem pochyłym zaznaczono komentarz.

Pozostała jeszcze do omówienia dyrektywa "ORG", której zadaniem jest deklarowanie adresu w pamieci programu procesora, od którego będą umieszczane kolejne, wystepujace po tej dyrektywie, bajty programu. I tak np. kiedy piszemy aplikacje na nasz komputerek edukacyjny, liczymy się z tym, że fizyczna pamieć (SRAM) do której ładowany jest program zaczyna się od adresu 8000h. Dlatego każdy z przykładów do wklepania zaczyna sie od dyrektywy:

```
ORG   8000h
```

co mówi kompilatorowi PASM51.EXE, że występujące instrukcje po tej dyrektywie ma umieszczać od adresu 8000h począwszy, czyli od początku fizycznej pamięci SRAM komputerka.

Dyrektywa ORG, tak jak omówione wcześniej, może być używana wielokrotnie w tym samym kodzie źródłowym programu. Ważne jest aby "panować"na nią, ale o tego nauczymy sie przy innej okazji, kiedy zajdzie taka potrzeba.

No i ostatnia sprawa dotycząca stosowania etykiet przy skokach warunkowych ale i bezwarunkowych,czy deklaracjach podprogramów. Dla porządku należy powiedieć że wszystkie etykiety powinny zaczynać się od pierwszej kolumny w danej linii tekstu, czyli od pierwszego znaku, dodatkowo należy je zakończyć znakiem dwukropka, np.

```
etyk01:
etyk02:
dodaj:
przyklad:
```

Wielkość liter z kodzie źródłowym programu w przypadku kompilatora PASM51.EXE nie ma znaczenia, wyjątek stanowią teksty będace argumentami instrukcji czy dane w tablicach tworzonych z wykorzystaniem dyrektywy "DB" i umieszczone pomiedzy znakami apostrofa, np. **'tekst'.**

Program monitora ("BIOS")

Ze względu na ograniczoną objętość a jednoczesnie dość szeroką tematykę związaną z programem monitora zawartym w EPROM'ie, a znajdującym się w każdym zestawie edukacyjnym AVT–2250, nie byłem w stanie opisać szczegółowo tego problemu przy okazji opisu kontrukcji i sposobów uruchomienia tego urządzenia w poprzednich numerach EdW.

Drugim powodem pewnego "ominięcia" tego tematu był fakt bezcelowości wyjasniania tego problemu na etapie kiedy nie znaliście jeszcze listy instrukcji procesora 8051, no przynajmniej większej jej części.

Skoro tak już sie stało i w ostatnich trzech numerach "wałkowaliśmy" teoretycznie ten temat, przyszła odpowiednia pora na wyjaśnienie sobie podstawowego pytania: "Co tak naprawdę siedzi w tym "monitorze" prócz funkcji usługowych które już znacie – czyli operacji umożliwiających np. ładowanie (LOAD), podglądanie (EDIT) i uruchamianie programów (JUMP)".

Jak wspomaniałem w lekcji nr 4 zamieszczonej w poprzednim styczniowym numerze EdW, w programie "monitora" znajdują się dodatkowe procedury – podprogramy (użyte określenia są tożsame), dzięki którym w prosty sposób możliwe jest wywołanie określonego działania naszego komputerka z zależności od Twoich potrzeb, czyli np. wyczyszczenie pola odczytowego wyświetlacza, odczyt nacisniętego klawisza, czy wyświetlenie liczby lub pseudo–tekstu w celu poinformowania uzytkownika o zaistniałym zdarzeniu lub wykonaniu pewnej operacji przez nasze urządzenie. W tym miejscu szczególnie "komputerowcy" stwierdzą że sprawa wygląda podobnie jak w przypadku zwykłego PC-ta czy każdego innego komputera, który posiada przecież dwa podstawowe urządzenia wejścia–wyjścia czyli klawiaturę i monitor (u nas jest to ośmiopozycyjny wyświetlacz 7–segmentowy).

Tym osobom chcę wyjaśnić, że aby zrozumieć znaczenie dodatkowych procedur o których mówię, wystarczy porównać "monitor"– program komputerka do coraz rzadziej ("niestety" z punktu widzenia elektronika–"hardware'owca" a zarazem programisty) dziś spotykanego systemu operacyjnego MS–DOS w waszym komputerze. Taki system oprócz tego że posiada pewne standardowe polecenia, chociażby do pokazania na ekranie drzewa katalogów na danym napędzie dyskietek lub dysku twardym ("dir" w PC-tach). Te polecenia można właśnie porównać do poleceń dostępnych bezpośrednio (bez pisania programu) z klawiatury komputerka edukacyjnego. W komputerze (np. PC) trzeba wpisać polecenie w postaci wyrazu i potwierdzić, u nas w naszym urządzeniu wystarczy nacisnąć jeden klawisz aby wywołać określoną funkcję – porównanie to dotyczy samej zasady posługiwana się programem monitora jako takim właśnie prościutkim systemem operacyjnym, który nagminnie nazywam "monitorem" lub "biosem".

Jak jednak wiecie (zwracam się nie tylko do komputerowców ale i do ciekawskich "ręczniaków"), może nie wszyscy, w każdym systemie operacyjnym istnieją dodatkowe funkcje usługowe, które wykorzystywane są poprzez różne programy, które użytkownik uruchamia z poziomu systemu np. gry, arkusze kalkulacyjne, edytory tekstów, lub inne specjalistyczne oprogramowanie, praktycznie wszystko. Takie funkcje np. w sysytemie MS–DOS dostepne są poprzez funkcję systemową INT21h. Programiści piszący aplikacje np,. w asemblerze x86 lub popularnym Turbo Pascalu z pewnością wiedzą o co chodzi.

Takie też funkcje posiada nasz komputerek,oczywiscie ze względu na ograniczone możliwości i potrzeby naszego systemiku oraz zupełnie odmienną architekturę procesora, są one mocno ograniczone, lecz dla naszych zastosowań w zupełności wystarczą.

Przejdźmy zatem do ich omówienia. Przestawiając wspomniane podprogramy będę kierował sie pewnym schematem, co z pewnoscia ułatwi zrozumienie poszczególnych procedur i pozwoli na ich bezproblemowe użycie w programie przykładowym, który przeanalizujemy krok po kroku w kolejnej lekcji nr 5. Oto schemat:

– umowna nazwa podprogramu oraz jego adres fizyczny (adres wywołania instrukcją LCALL)
– krótki opis działania danego podprogramu, "czyli co w efekcie się stanie"
– parametry wejściowe ("we:") wywołania, czyli "co i do jakiego rejestru trzeba załadować przed wywołaniem podprogramu
– parametry wyjściowe ("wy:"), czyli wynik (efekt) działania danej procedury po jej zakończeniu
– dodatkowo podam w niektórych przypadkach, "co taki podprogram zmienia (jakie rejestry) w procesorze". Ta informacja jest bardzo ważna, bowiem przecież pisząc jakiś program operujemy na rejestrach procesora, do których zapisujemy określone wartości, i w których przechowuujemy wyniki operacji. Niektóre z tych rejestrów (np. akumulator lub rejestr B, czy rejestry R0...R7) są także wykorzystywane dodatkowo w celu zrealizowania określonej operacji (wewnątrz podprogramu) a więc ich zawartość jest w wyniku działania danej procedury zmieniana. Konieczne zatem się staje zapamiętanie przechowanie na czas wykonywania podprogramu zawartości tych rejestrów, aby po zakończeniu procedury, mozna było odtworzyć ich zawartość pierwotną. Ktoś w tym miejscy moze zadac pytanie: "a dlaczego każdy podprogram po rozpoczęciu swego działania (po wywołaniu) sam nie zadba i nie przechowa tych rejestrów, po czym je odtworzy po zakończeniu działania – przeciez jest to możliwe...". Tak ale z punktu widzenia użytkowego, taka sytuacja jest w praktyce po prostu niepotrzebna, wydłuża to tylko niepotrzebnie działanie podprogramu, a co za tym idzie spowalnia działanie programu głównego.

HEXASCII (0235h)

Przejdźmy zatem do opisu poszczególnych procedur – podprogramów.

– zamienia liczbę znajdującą się w akumulatorze (Acc) na dwa znaki ASCII których kody umieszcza w rejestrach B (starszy półbajt) i A (młodszy półbajt), np. jeżeli w Acc przed wywołanie podprogramu bedzie liczba 4Fh (wszystkie liczby w kodzie heksadecymalnym), to po zakończeniu wykonywania procedury będzie: B = '4` (52) a akumulator Acc= F (70)
– adres wywołania: 0235h
– we: Acc – 8–bitowa liczba do zamiany
– wy: B – kod starszego półbajtu liczby wejściowej, Acc – to samo ale młodszy
– przykład:

```
MOV   A, #60h        ;liczba 60h do zamiany
LCALL HEXASCII       ;wywołanie podprogramu
.......              ;po wykonaniu w B będzie
                     znak '6  (liczba 54)
.......              ;a w Acc kod znaku '0` (liczba 48)
```

Procedurę można też wywołać w postaci :

```
LCALL 0235h
```

co przyniesie taki sam skutek. "Ręczniacy" mogą sobie od ręki przetłumaczyć na język maszynowy te polecenie jako ciąg bajtów: " 02, 02, 35h " – pierwszy bajt to kod instrukcji LCALL, dwa następne to adres procedury HEXASCII. Tak samo można postępować z każdym innym wywołaniem, jednak w przypadku "komputerowców" ze względu na czytelność programu, należy używać nazw słownych.

A2HEX (024Eh)

– powoduje wyświetlenie zawartości akumulatora na wyświetlaczu w postaci dwóch znaków ze zbioru 0...9, A...F, na pozycji (wyświetlaczu) podanej w rejestrze B
– adres wywołania: 024Eh
– we: Acc – 8–bitowa liczba do wyświetlenia, B – numer wyświetlacza (1...8) od którego ma sie zaczynać wyświetlana liczba
– wy: wyświetlenie liczby na określonej pozycji
– traci: R0, A i B
– przykład: patrz lekcja nr 4 z poprzedniego numeru EdW, inny przykład:

;niech w rejestrze pod nazwą "sek" bedą przechowywane sekundy programu zegara, który akurat uruchomilismy w naszym komputerku za pomoca innego podprogramu, w rej. "min" minuty a w rejestrze "godz" – godziny aktualnego czasu, aby pokazać aktualny czas w postaci np.

GG_ MM_ SS ,gdzie GG – pozycja godzin,
;MM – pozycja minut
;SS – pozycja sekund

(podkreślenie oznacza wygaszoną pozycję wyświetlacza)
np. "12 45 00" (godz 12:45 i 00 sekund) nalezy wykonac sekwencję:

```
LCALL CLS       ;najpierw wyczyścimy całe pole odczytowe
MOV   B, #1     ;godziny wyświetlimy od 1 wyświetlacza
MOV   A, godz   ;załadowanie godzin
LCALL A2HEX     ;i wywołanie podprogramu
                ;...... po wykonaniu na DL1 i DL2 będzie
                godzina
MOV   B, #4     ;minuty wyświetlimy od 4 wyświetlacza
MOV   A, min    ;załadowanie minut
LCALL A2HEX     ;i wywołanie podprogramu
                ;...... po wykonaniu na DL4 i DL5 będą
                minuty
MOV   B, #7     ;sekundy wyświetlimy od 7 wyświetlacza
MOV   A, sek    ;załadowanie sekund
LCALL A2HEX     ;i wywołanie podprogramu
                ;...... po wykonaniu na DL7 i DL8 będą
                sekundy
```

DPTR4HEX (025Fh)

– wyświetla 16–bitową liczbę zawartą w parze rejestrów DPH i DPL (DPTR) jako 4 znaki 0...9, A...F na pozycji podanej w rejestrze B (1...5).
– adres wywołania: 025Fh
– we: DPTR – liczba do wyświetlenia, B – numer pozycji na wyświetlaczu (1...5)
– wy: wyświetla na 4 pozycjach 16–bitowa liczbę z DPTR
– traci: R0, A i B
– przykład:

```
MOV   DPTR,#1998h   ;wyświetl aktualny rok
MOV   B, #3         ;od 3–ciego wyświetlacza
LCALL DPTR4HEX      ;wywołanie podprogramu
                    ;po wykonaiu na DL3–4–5–6 będzie
                    wyświetlony;rok z DPTR
```

CLS (0274h)

– czyści całe pole wyswietlacza (wstawia spacje na pozycje DL1...8)
– adres wywołania: 0274h
– we: bez parametrów
– wy: czyści wyswietlacz
– traci: R0 i R7 z aktualnie ustawionego zbioru
– przykład:

```
LCALL CLS           ;wyczyszczenie wyświetlacza
```

Sławomir Surowiński

Lekcja 5

W dzisiejszej lekcji przeanalizujemy obszerniejszy przykład programu, którego zadaniem będzie realizacja funkcji prostego minutnika ze sterowaniem dowolnego urządzenia zewnętrznego poprzez wybrane końcówki portu P1 mikroprocesora. W programie tym wykorzystamy omówione w poprzednich lekcjach instrukcje procesora 8051 oraz niektóre procedury usługowe czyli podprogramy zawarte w programie monitora znajdującego się w EPROMie waszego komputerka edukacyjnego.

Na listingu poniżej znajduje się wspomniany program w postaci generowanej przez kompilator PASM51 znajdujący się na dyskietce AVT-2250/D. Przypomnę tylko ze właściwy kod źródłowy programu zaczyna się od trzeciej kolumny tekstu (nie wliczając numerów linii, podanych w nawiasach). Pierwsza kolumna zawiera zapisany w postaci szesnastkowej adres pod którym występuje znajdująca się w danej linii instrukcja. W przypadku kiedy w linii nie występuje instrukcja lecz deklaracja przypisania EQU liczba ta wskazuje na argument znajdujący się po prawej stronie.

W drugiej kolumnie jak zapewne zdążyliście się zorientować z poprzednich przykładów, znajduje się przetłumaczony na kod maszynowy ciąg 8-mio bitowych licz zapisanych także w postaci heksadecymalnej. Dalsza cześć linii tak jak wspomniałem przed chwila, to kod źródłowy programu.

Dla ułatwienia dodatkowo wszystkie linie ponumerowano liczbami w nawiasach.

Przed stworzeniem dowolnego programu, każdy programista powinien zastanowić się nad tym co powinien on realizować – jaka ma być jego funkcja?. W naszym przypadku program, jak powiedziałem wcześniej, ma być minutnikiem, czyli mówiąc inaczej timerem z możliwością wprowadzenia przez użytkownika czasu, a następnie załączenie urządzenia zewnętrznego na ten wprowadzony wcześniej okres. Zakładamy, że czas będzie ustawiany w minutach i sekundach, a maksymalny zakres ustawionego czasu to 99 minut i 99 sekund. Dodatkowo wykorzystamy wyświetlacz naszego komputerka oraz jego klawiaturę do wprowadzenia nastaw minut i sekund oraz wyświetlenia go w trakcie odliczania aż do osiągnięcia stanu 0:00 i wyłączenia sterowanego urządzenia.

Przed rozpoczęciem pisania programu określmy sobie scenariusz, który często odpowiada tzw. algorytmowi programu. Ponieważ nasz program jest dość prosty, taki algorytm można zapisać w postaci kolejnych podpunktów, oto one.

a) wprowadzenie z klawiatury minut
b) wprowadzenie z klawiatury sekund – mamy czas załączenia
c) wybór przez użytkownika końcówki procesora (np. z portu P1) sterującej urządzeniem
d) oczekiwanie na rozpoczęcie odliczania. Zakładamy że polaryzacja sygnału sterującego będzie ujemna.
e) po naciśnięciu odpowiedniego klawisza (umownie nazwanego jako START) przez operatora timer zaczyna odliczanie w tył do zera z wyświetleniem pozostałego czasu. Załóżmy że w tym miejscu programu istnieje (po wciśnięciu innego klawisza) dodatkowa możliwość ustawienia innego czasu, np. w przypadku kiedy pomyliliśmy się przy wpisywaniu czasu włączenia.
f) po zakończeniu odliczania urządzenie powinno zostać wyłączone, a program powinien powrócić do punktu e) dając dwie możliwości:
– rozpoczęcia ponownego odliczania po wciśnięciu klawisza START lub
– powrotu do punktu a) celem poprawienia (zmiany) nastaw minut i sekund.

Pamiętajmy, że nasz program napisany będzie z wykorzystaniem podprogramów – procedur standardowych naszego monitora i w związku z tym jego działanie będzie możliwe tylko z wykorzystaniem komputerka AVT-2250 z zawartym w EPROM ie monitorem.

W podanym niżej przykładzie wykorzystane będą pewne, nie omówione wcześniej rejestry procesora, które modyfikowane są automatycznie przez cały czas działania komputerka. Modyfikacje te odbywają się „w tle”, tzn. że pogram monitora jest skonstruowany tak, że nawet podczas działania naszego programu z przykładu, wspomniane modyfikacje rejestrów odbywają się nieprzerwanie. Dzieje się tak za sprawą układu licznikowego procesora oraz systemu przerwań. Sprawa jest nieco skomplikowana, a ponieważ będzie omówiona w kolejnych lekcjach, nie będziemy jej teraz szczegółowo omawiać.

Omówimy sobie teraz tylko krótko wspomniane rejestry, tak aby lepiej zrozumieć działanie programu.

Otóż program monitora do działania potrzebuje kilku rejestrów, które fizycznie znajdują się w obszarze wewnętrznej pamięci RAM procesora, przeznaczonej do dowolnego wykorzystania przez użytkownika. Oto kilka z nich, wykorzystane w naszym programiku:

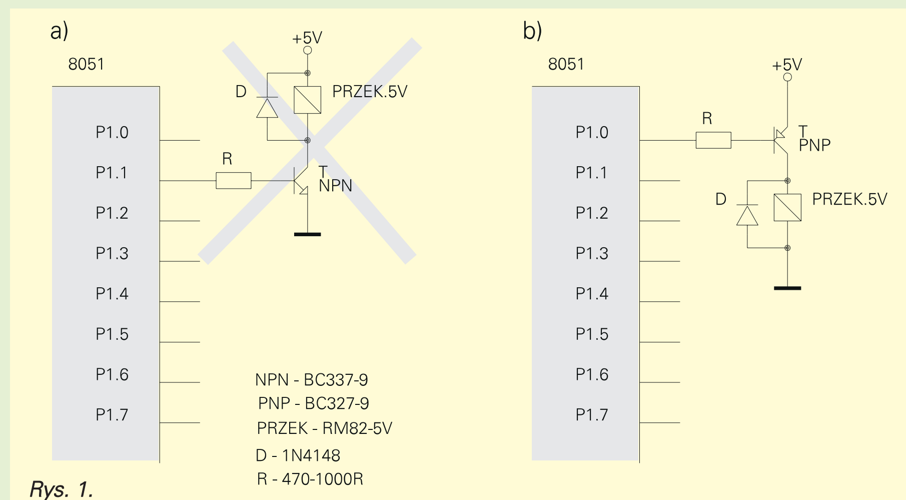
DL1...DL8 (adresy: 78h...7Fh) – osiem rejestrów, które pełnią rolę bufora znaków wyświetlanych przez 7-segmentowy wyświetlacz naszego komputerka. Zapis liczby odpowiadającej wyświetlanemu znakowi do jednego z tych rejestrów powoduje zapalenie odpowiednio jednego z 8-miu pozycji wyświetlacza. Wyświetlane znaki tworzone są następująco. Każdy z ośmiu bitów liczby danego znaku odpowiada poszczególnym segmentom wyświetlacza. Bitów w liczbie 8-bitowej jest oczywiście osiem, a segmentów w pojedynczym wyświetlaczu także osiem wliczając w to kropkę. Układ elektroniczny komputerka jest tak skonstruowany (przypomnij sobie opis z EdW 9/97), że ustawienie bitu (1) w tej liczbie powoduje zapalenie danego segmentu, odwrotnie wyzerowanie (0) zgaszenie go. I tak najstarszy bit 7 odpowiada za segment H – czyli kropkę, najmłodszy bit 0 za segment A wyświetlacza w kolejności jak na **rysunku 1**.

Dla przykładu podam kilka znaków i odpowiadające im liczby (kody).

Bity: 7-6-5-4-3-2-1-0
Segmenty: H G F E D C B A (msb...lsb)

Liczba	Binarnie	Znak
63	00111111	– cyfra ‘0’
6	00000110	– cyfra ‘1’
91	01011011	– cyfra ‘2’
127	01111111	– cyfra ‘8’
111	01101111	– cyfra ‘9’
119	01110111	– litera ‘A’
124	01111100	– litera ‘B’
94	01011110	– litera ‘D’
64	01000000	– znak ‘-’ (minus)
118	01110110	– litera ‘H’

Zastanów się chwilę i pomyśl jak można utworzyć inne znaki, np. „L”, „4”, „7”. Posiadacze dyskietek AVT-2250/D mogą znaleźć od-



Rys. 1.

powiedź w pliku CONST.INC, gdzie zdefiniowano większość znaków używanych w naszych aplikacjach. Pozostałe osoby będą miały okazję zapoznać się z tymi informacjami w kolejnej lekcji szkoły mikroprocesorowej.

BSW (adres: 70h) – Kolejnym istotnym rejestrem jest nazwane umownie „słowo stanu monitora". Każdy z bitów tego słowa pełni określona rolę. I tak np. najmłodszy bit 0 zmienia swój stan (z „0" na „1" i odwrotnie) z częstotliwością 2 Hz, co będzie wykorzystane w naszym programie do odliczania sekund. Dwukrotna zmiana bitu może bowiem być informacją, że upłynęła właśnie dokładnie 1 sekunda.

BLINKS (adres: 71h) – bajt, którego poszczególne bity 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 i oczywiście można ją zmieniać, lecz sposób tej sposób na to nie jest tematem tego artykułu. 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

Teraz zajmijmy się już naszym programem.

Na początku znajduje się deklaracja CPU procesora oraz kilka linii komentarza, które w każdym przypadku zaczynają się znakiem średnika.

W linii (1) zdefiniowano dodatkowy znak imitujący małą literę „o", w naszym programie minutnika będzie on zapalany na ostatniej pozycji celem sygnalizacji że urządzenie jest załączone i odliczany jest ustawiony wcześniej czas.

Linie (2) i (3) to definicje dodatkowych rejestrów w wew. RAM procesora, w których będą przechowywane wprowadzone i potem dekrementowane minuty i sekundy ustawionego czasu minutnika.

Linia (4) to deklaracja dla kompilatora, który musi wiedzieć że program ma zaczynać się od adresu 8000h, gdzie zlokalizowana jest pamięci operacyjne SRAM naszego komputerka i gdzie będzie załadowany nasz program.

Od komentarza przed linią (5) zaczyna się właściwy program. Na początku trzeba wyczyścić wyświetlacz wywołując podprogram nazwany jako CLS (jego adres w pamięci monitora to 0274h). Dodatkowo linia ta została oznaczona etykietą „pocz:" która posłuży potem do powrotu na początek programu po zakończeniu jednego cyklu odliczania i chęci rozpoczęcia następnego.

Linia (6) powoduje ustawienie wszystkich końcówek portu P1 w stan wysoki, co w naszym załączeniu oznacza wyłączone urządzenie zewnętrzne. Komuś w tym miejscu może wydawać się, że bardziej logiczne byłoby wyzerowanie końcówek i sterowanie np. przekaźnikiem za pomocą tranzystora NPN (jak pokazano na rys.1a) lecz takie rozumowanie może w pewnych przypadkach być błędne. Jak wiadomo w prawdziwych zastosowaniach wszelkie stany przejściowe szczególnie układów wykonawczych są niepożądane. Tak tez może się zdarzyć w przypadku sterowania przekaźnika z rys.1a, bowiem procesor po włączeniu zasilania przez krótką chwilę (kilka do kilkuset milisekund) znajduje się w stanie resetu, a końcówki jego portów znajdują się w stanie wysokiej impedancji. W takim przypadku sterowany przekaźnik w sposób jak na rys.1a zostałby załączony na krótko po włączeniu zasilania, układu, aż do momentu planowanego wyłączenia na początku programu instrukcją np. MOV P1, #0 – etykieta „pocz:".

Linie (7)..(12) powodują wyświetlenie znaków informacyjnych, będących zachętą do wprowadzenia danych przez użytkownika przed rozpoczęciem odliczania.

"– – – – –F"

W linii (13) znajduje się wywołanie podprogramu WAITSEK, który nie ma specjalnego przeznaczenia, poza tym że wykonanie go zajmuje procesorowi około 1 sekundy. Procedura ta zdefiniowana jest w dalszej części listingu, toteż omówimy ją za chwilę.

W kolejnej części programu następuje pobranie liczby minut za pomocą podprogramu GETACC (adres: 03A7h) – linia (16) na pozycji podanej w linii (15) w rejestrze B przed wywołaniem podprogramu. Dodatkowym użytecznym „wodotryskiem" są migające pozycje

1 i 2 wyświetlacza, co zachęca użytkownika do wprowadzenia liczby minut.

W linii (17) wprowadzona liczba jest zapamiętywana w zdefiniowanym wcześniej rejestrze nazwanym jako „minuty".

Warto w tym miejscu powiedzieć, że liczbę minut (a także sekund) należy wprowadzać korzystając z klawiszy cyfr: 0...9 – czyli w postaci dziesiętnej, łatwiejszej dla przeciętnego użytkownika. Podprogram GETACC akceptuje oczywiście także klawisze A...F, lecz nie należy ich używać, bowiem spowoduje to nieprawidłową prace programu, dlaczego o tym za chwilę.

Po pobraniu minut w liniach (19)...(22) realizowane jest w analogiczny sposób wprowadzenie sekund, z tą różnicą, że tym razem migoczą pozycje 4 i 5 wyświetlacza, co jest naturalne i czytelne dla operatora.

Linie (23)...(25) realizują pobranie numeru aktywnego wyjścia portu P1, do którego końcówki dołączone jest sterowane zewnętrznym urządzeniem. Numer wyjścia powinien być podany za pomocą klawiszy od: „0" (dla końcówki P1.0) do „7" (dla końcówki P1.7).

Wykorzystany zostaje do tego inny podprogram monitora: „GETDIGIT" – pobranie cyfry szesnastkowej za pomocą klawiszy 0...9, A...F. Po wykonaniu tej operacji (podobnie jak w przypadku GETACC) cyfra znajduje się w akumulatorze. Rejestr B przed wywołaniem powinien zawierać pozycję na której będzie wyświetlona wprowadzana przez operatora cyfra.

W linii (26) cyfra ta jest dodatkowo „filtrowana", tak że pozostają 3 najmłodsze bity – czyli cyfra z zakresu 0...7. Linia ta jest w zasadzie nie jest konieczna, lecz pokazuje w jak prosty sposób można czasem zabezpieczyć się przed nieprawidłowym wprowadzeniem danych.

Następnie w linii (27) numer wyjścia zapamiętany zostaje w rejestrze B, gdzie poczeka na „swój moment".

W linii (28) kończy się wprowadzanie danych, a displej wyświetlając na przedostatniej pozycji migający znak „-" informuje użytkownika o gotowości do rozpoczęcia odliczania.

W linii (29) znajduje się wywołanie podprogramu, którego zadaniem jest oczekiwanie na naciśnięcie dowolnego klawisza a następnie zwrócenie jego kodu w akumulatorze. W przypadku klawiszy 0...9, A...F kody te odpowiadają kodom ASCII przyjętym na świecie, jedynie naciśnięcie klawisza OK. ("okej") powoduje zwrot kodu 13.

W linii (30) sprawdzany jest warunek, czy kod ten odpowiada naciśnięciu klawisza START, czyli OK. Jeżeli tak, to wykonane zostaną linie programu począwszy od linii (31).

Linia (31) — wykonywana jest w po uruchomieniu odliczania. Wywołana w niej procedura WAITSEK powoduje odczekanie okresu dokładnie 1 sek., tak aby przygotować odliczanie a jednocześnie spowolnić działanie programu po wciśnięciu klawisza OK.

W liniach (32) i (33) wykonywana jest „kosmetyka" wyświetlacza, polegająca na zapaleniu migającej kreski rozdzielającej minuty i sekundy, co daje efekt wizualny upływu czasu.

Następnie w linii (34) pobrany zostaje adres tabeli bajtów z których każdy po zapisaniu do portu P1 (linia 37) spowoduje załączenie jednego z 8-miu wyjść. I tak w linii (35) zostaje odtworzony w akumulatorze a przechowywany w rejestrze B (patrz linia nr 27) numer wybranego wyjścia portu P1.

W linii (36) pobrany zostaje bajt z tabeli zdefiniowanej w dalszej części kodu programu, a w następnej bajt ten zostaje wpisany do portu P1 załączając jedno z 8-miu wyjść. Popatrzmy przez chwilę i zastanówmy się, jak to się dzieje. Otóż wyobraźmy sobie że podczas wprowadzania numeru aktywnego wyjścia wcisnęliśmy (za pomocą procedury GETDIGIT, linia 25) klawisz 4, czyli w domyśle wybraliśmy wyjście P1.4, gdzie ma wystąpić założony wcześniej stan aktywny (u nas niski), a na pozostałych końcówkach stan nieaktywny (u nas wysoki). Teraz jeżeli pobierzemy za pomocą instrukcji MOVC A, @A+DPTR, ze zdefiniowanej tabeli, bajt z przesunięciem 4 względem początku tabeli, to fizycznie będzie to 5-ty bajt – popatrz na tabelę! Pierwszy bajt w tabeli ma przesunięcie 0 bo jest on umieszczony pod adresem umieszczonym w rejestrze DPTR. Adres tabeli został przecież wpisany w linii (34). Drugi bajt będzie miał przesunięcie 1, trzeci – 2 itd.

Popatrzmy teraz na sam bajt, pobrany przed chwilą ze zdefiniowanej tabeli wyjść. Piąty z tabeli bajt to "11101111b", po wpisaniu w linii 37 na wszystkich końcówkach oprócz P1.4 pojawi się logiczna 1-ka, a na tym jednym stan niski, proste prawda.

W linii (38) dodatkow po załączeniu wyjścia, zapalony zostaje znaczek "o" na ostatniej pozycji wyświetlacza, co sygnalizuje operatorowi, że wyjście zostało uaktywnione, a przyłączone do niego urządzenie pracuje.

Teraz następuje część programu, w której ustawiony czas będzie odliczany aż do zera. Do tego celu wykorzystane zostają dwa rejestry robocze R3 i R4. Do nich to właśnie zostaje przepisana zawartość rejestrów "minuty" i "sekundy". Taka operacja jest potrzebna do tego, aby w czasie odliczania nie zmniejszane były rejestry przechowujące ustawiony czas ("minuty" i "sekundy"), tylko inne wolne rejestry, u nas R3 i R4. Dzięki temu kiedy czas osiągnie zero, a użytkownik będzie chciał powtórzyć proces odliczania, będzie mógł to zrobić bez ponownego ustawiania nastaw minut i sekund, powtórnie przepisując nastawy do rejestrów R3 i R4.

W liniach (42) i (43) sprawdzany jest warunek, czy nastawa sekund jest równa 0?. Wykorzystana tu jest instrukcja JNZ, po uprzednim przepisaniu R4 do akumulatora.

W tym miejscu zamiast tych dwóch instrukcji można by użyć polecenia

CJNE R4, #0, niezer,

ale wtedy między etykietę "niezer" a instrukcję kall DECACC trzeba wstawić instrukcję: MOV A; R4" zastanów się jednak dlaczego? Jeżeli warunek jest spełniony, czyli zawartość licznika sekund (rejestru R4) jest różna od zera, następuje skok do miejsca programu oznaczonego przez etykietę "niezer".

Tutaj w linii (53) wywołany zostaje podprogram, którego zadaniem jest dekrementacja zawartości akumulatora, który przecież zawiera to co przed chwilą było w R4, czyli licznik sekund. Dlaczego użyto podprogramu, a nie np. instrukcji DEC A, która przecież także zmniejsza zawartość rejestru (akumulatora), a no dlatego, że instrukcja ta dekrementuje rejestr w sposób naturalny w kodzie szesnastkowym, czyli np.
59, 58, 57, 56, 55, 54, 53, 52, 51, 50, jak dotąd jest wszystko w porządku, ale następne zmniejszenie rejestru nie spowoduje ustawienia sekund na liczbę 49 ale na 4Fh !, a tego nie chcielibyśmy wyświetlić na displeju jako pozostałych sekund. Gdyby zatem użyć tej pojedynczej instrukcji, to nie tylko odczyty na wyświetlaczu byłyby bezsensowne, ale i 10 sekund w naszym timerku trwałoby aż 16 sekund!

Dlatego od linii (79) rozpoczyna się zdefiniowany podprogram – procedura, której zadaniem jest właściwe, dziesiętne (lub jak kto woli z korekcją dziesiętną) zmniejszenie akumulatora, czyli w naszym przypadku jeżeli np. licznik sekund będzie zawierał 50, to po dekrementacji za pomocą naszego podprogramu będzie w nim liczba 49, a więc to co trzeba. Ta sama sprawa dotyczy licznika minut, ale o tym za chwilę.

Tak więc po zmniejszeniu, dalej w linii (54), zdekrementowana zawartość akumulatora zostaje przepisana do właściwego rejestru R4, po czym w liniach następnych (55 — oznaczona jako “wypisz”) i (56), następuje wyświetlenie aktualnej zawartości sekund na displeju. Korzystamy tu z podprogramu A2HEX, omówionego w tym odcinku szkoły mikroprocesorowej. W linii (57) następuje planowe odliczenie czasu 1 sekundy z wykorzystaniem podprogramu WAITSEK, a następnie skok z powrotem do momentu kiedy sprawdzany jest warunek wyzerowania sekund — etykieta “nsek” (linia 42).

Wróćmy teraz do linii (43), kiedy to warunek zgodności sekund z 0 jest spełniony. Wtedy wykonana zostanie instrukcja następna po “JNZ niezer”, czyli linia (44).

W niej to do akumulatora zostaje załadowana zawartość rejestru R3, która jest przecież odzwierciedleniem licznika minut.

W linii (45) zostaje sprawdzony warunek, czy licznik minut (przepisany przed chwilą do akumulatora) jest równy zero. Jeżeli tak jest to znaczy że licznik timera do szedł do zera i nastąpi skok do etykiety “koniec:” - linia 59, którą omówimy za chwilę.

Jeżeli licznik minut jest różny od zera, to wykonana zostanie linia (46), czyli dekrementacja minut (przepisanych przecież do akumulatora) za pomocą omówionego wcześniej podprogramu DECACC.

Po tym w linii (47) zmniejszona w A liczba minut zostaje przepisana do rejestru R3, a w kolejnych liniach (48) i (49) wypisana na pozycji wyświetlacza określonej w rejestrze B (linia 48).

Teraz uwaga, skoro zmniejszono liczbę minut, to teraz w linii (50) zostaje ustawiona początkowa (przy zmniejszaniu) liczba sekund, czyli 59. No bo skoro było np.

19:00 (min:sek)

to po zmniejszeniu o 1 sekundę licznik powinien wskazywać

18:59 (min:sek)

tak też się stanie, dzięki tej operacji. Oczywiście skorygowana tutaj liczba sekund musi być zapisana w rejestrze R4, co dzieje się w linii (51).

Teraz należałoby wypisać na displeju nowe wartości licznika timera, toteż w kolejnej linii (52) znajduje się skok bezpośredni do miejsca znanego nam już, czyli do etykiety ”wypisz”.

Po tym i wykonaniu linii (57) i (58), które opisałem wcześniej, program skacze na początek pętli zmniejszania licznika timera, czyli do etykiety “nsek” — linia 42 i pętla powtarza się.

Takie odliczanie w tył licznika timera odbywa się, jak zapewne zauważyliście, aż do momentu, gdy nastąpi sytuacja, gdy minuty jak i sekundy będą równe zero. Wtedy warunek w omówionej wcześniej linii (43) nie będzie spełniony (sekundy = 0), a warunek w linii (45) spełniony (minuty = 0) i nastąpi skok do etykiety “koniec:” - linia 59.

Tutaj do portu P1 zostaje zapisana liczba 255, binarnie 11111111, czyli wszystkie końcówki portu zostają dezaktywowane, a przy tym zostaje wyłączone sterowane urządzenie.

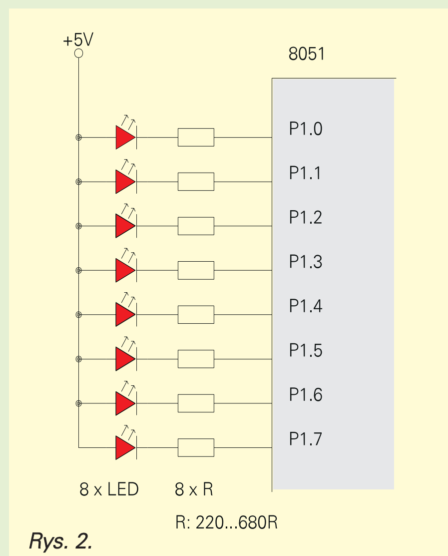
W linii (60) wyłączone zostaje migotanie niektórych zapalonych wcześniej znaków na wyświetlaczu. W linii kolejnej na pozycji 8 zostaje wypisany znak “F” co jak wspomniałem na początku programu, jest sygnałem dla użytkownika, że urządzenie zostało wyłączone.

W liniach (62)....(67) program wypisuje ustawioną wcześniej, nastawę minut i sekund timera, były one przez cały czas przechowywane w rejestrach nazwanych jako “minuty” i “sekundy”. Dzięki użyciu rej. roboczych R3 i R4 jak powiedziałem wcześniej ich zawartość nie uległa zmianie podczas dekrementacji.

Następnie w linii (68) następuje skok do etykiety “wait”, gdzie zgodnie z naszym scenariuszem, program czeka na kolejną decyzję operatora, który ma do wyboru dwie sytuacje. Pierwsza to rozpoczęcie ponownego odliczania ustawionych nastaw minut i sekund (klawisz OK.), druga to ustawienie nowych - klawisz “0” (zero).

Program wraca zatem do linii (29), wróćmy i my w naszej analizie. Tutaj jak pamiętamy wywoływany jest podprogram CONN, którego zadaniem jest czekanie na naciśnięcie klawisza i zwrócenie jego kodu w akumulatorze.

W poprzednim przypadku po ustawieniu timera wcisnęliśmy klawisz OK. i program potoczył się jak opisywałem wcześniej.



Rys. 2.

Załóżmy teraz że wciśniętym klawiszem nie był OK., toteż warunek w linii (30) będzie spełniony i nastąpi skok do etykiety “nieok” — linia 69. Tutaj kod wciśniętego klawisza jest porównywany z liczbą — kodem “0”, czyli poleceniem zmiany nastaw timera (minut i sekund). Jeżeli nie wciśnięto klawisza “0”, to warunek będzie spełniony i nastąpi skok do linii (29), czyli do ponownego oczekiwania na wciśnięcie klawisza.

Jeżeli natomiast operator wciśnie “0”, warunek w linii (69) nie będzie spełniony i program przejdzie do linii (70), gdzie występuje polecenie skoku bezwarunkowego na początek programu. Tam w linii (5) wszystko zaczyna się od początku.

Tak moi drodzy i tak działa cały program z naszego przykładu. Ktoś może zadać pytanie, a co zrobić jak chcę zakończyć program, nic prostszego. Przypominam że do “eleganckiego” wyjścia z prawie każdego programu służy klawisz monitora “M.”. Można także zresetować komputerek, i potem uruchomić program od nowa.

Pozostały do omówienia dwa zaimplementowane w programie procedury: WAITSEK — linia 71, i DECACC — linia 79.

Pierwsza procedura WAITSEK opiera się na zasadzie sprawdzenia dwukrotnej zmiany stanu najmłodszego bitu w słowie specjalnym monitora BSW, opisanym na początku lekcj. Dwukrotna zmiana fazy (stanu) tej flagi wyznacza nam okres 1 sekundy, przy założeniu, że podprogram jest wykonywany cyklicznie. Zastanów się dlaczego np. przy pierwszym wywołaniu tej procedury okres oczekiwania może być mniejszy? — odpowiedź w kolejnej lekcji naszej klasy mikroprocesorowej.

Dodatkowo ze względu na to że w podprogramie wykorzystywany jest i modyfikowany akumulator, w linii (71) następuje zapamiętanie stanu tego rejestru na stosie, a po zakończeniu wykonywania operacji, w linii (77) odtworzenie jego stanu poprzez zdjęcie go ze stosu. Podprogram kończy się oczywiście, zgodnie z zasadami pisania procedur, instrukcją RET w linii (78).

Analizę i wyjaśnienie zasady działania podprogramu DECACC proponuję wykonać samodzielnie. Najlepiej jest za pomocą ołówka i kartki papieru, analizując linia po linii kod podprogramu, obliczać wartości akumulatora po wykonaniu każdej z linii procedury DECACC aż do instrukcji RET. Pamiętajmy tylko, że przy wywołaniu tego podprogramu argumentem jest liczba zawarta w akumulatorze, a po zakończeniu wynik także znajduje się w rejestrze A, lecz jest zmniejszony o 1 w sposób jaki omówiłem wcześniej.

Na koniec lekcji podaję kilka wskazówek, co można zmienić w programie, przynajmniej na platformie obsługi, tak aby zachować funkcjonalność programu. I tak:

– Można zmienić symbolikę wyświetlanych informacji, definiując swoje znaki zamiast symboli włączenia “o” lub wyłączenia “F” urządzenia.
– Ciekawą modyfikacją jest także zezwolenie na sterowanie kilkoma urządzeniami na raz w ośmiu dozwolonych kombinacjach. Można to uzyskać modyfikując poszczególne bity liczb umieszczonych w tabeli “tab_wyj”.
– Można także zmienić rodzaje klawiszy uruchamiających procedurę odliczania lub wprowadzania nowych nastaw przez opera-

tora oraz wprowadzić dodatkowy klawisz kończący program np. komunikatem "End".

Proponuję jako zadania wykonać następujące ćwiczenia.

Zadanie 1
Zaprojektować i użyć zamiast symboli "F" i "o" inne wskazujące na stan pracy sterowanego urządzenia.

Zadanie 2
Wprowadzić do programu obsługę i rozpoznawanie klawisza "1" jako kończącego program napisem "End". Nie zapomnijcie o pętli nieskończonej typu

stop: SJMP stop

po instrukcjach wypisujących wspomniany napis, inaczej program wpadnie "w maliny" i nic nie zobaczycie.

Zadanie 3
Zmodyfikować tabelę wyjść "tab_wyj", tak aby przy wymienionych poniżej nastawach wyboru wyjścia (0...7) możliwe były odpowiadające im, a wypisane obok kombinacje załączenia wyjśc portu P1. Dodatkowo proponuję do portu P1 dołączyć 8 diod LED z włączonymi w szereg opornikami (180...470 omów) i sprawdzić efekt swojej pracy, obserwując stan wyjść po uruchomieniu tajmera. Przypominam o prawidłowym włączeniu każdej z diod LED w sposób jak na **rysunku 2**.

Wszystkie odpowiedzi znajdziecie, drodzy Czytelnicy w kolejnej lekcji szkoły mikroprocesorowej. Życzę wiele cierpliwości oraz dużo satysfakcji!

Sławomir Surowiński

Listing programu

```
                        CPU       '8052.def'
                        ;Lekcja nr 5 szkoly mikroprocesorowej w EdW
                        ;program prostego minutnika z ustawianiem czasu
                        ;zalaczenia zewnêtrznego urz¹dzenia sterowanego
                        ;poprzez jedna z koncowek portu P1
                        ;-----------------------------------------------------------------------------
(1)   005C              _o                  equ     01011100b        ;definicja znaku "o" wyswietlacza
(2)   0050              minuty              equ     50h              ;rejestr przechowujacy minuty
(3)   0051              sekundy   equ       51h                      ;rejestr przechowujacy sekundy
                        ;------------------------------------------------------------------------------
(4)   8000                        org       8000h                    ;poczatek pamieci zewn.RAM

                        ;napisy i znaki informacyjne
(5)   8000 120274       pocz:     lcall     CLS                      ;wyczysc displej
(6)   8003 7590FF                 mov       P1,#255                  ;wszystkie piny portu OFF
(7)   8006 757840                 mov       DL1,#_minus
(8)   8009 757940                 mov       DL2,#_minus
(9)   800C 757B40                 mov       DL4,#_minus              ;napis "— —", zacheta do o
(10)  800F 757C40                 mov       DL5,#_minus              ;wprowadzenia czasu
(11)  8012 757E40                 mov       DL7,#_minus              ;i dodatkowo znacznik "-F"
(12)  8015 757F71                 mov       DL8,#_F                  ;wylaczenia "F" wyjscia "-"
(13)  8018 1280A4                 lcall     WAITSEK

                        ;wprowadzanie czasu - minuty
(14)  801B 757103                 mov       blinks,#00000011b        ;miga pozycja minut
(15)  801E 75F001                 mov       B,#1                     ;na pozycji nr 1 displeja
(16)  8021 1203A7                 lcall     GETACC                   ;pobranie liczby minut 0..99
(17)  8024 F550                   mov       minuty,A                 ;i zapamietanie jej w rejestrze
(18)  8026
                        ;wprowadzanie czasu - sekundy
(19)  8026 757118                 mov       blinks,#00011000b        ;miga pozycja sekund
(20)  8029 75F004                 mov       B,#4                     ;na pozycji nr 4 displeja
(21)  802C 1203A7                 lcall     GETACC                   ;pobranie liczby sekund 0..59
(22)  802F F551                   mov       sekundy,A                ;i zapamietanie jej w rejestrze

                        ;wybor wyjscia - pinu portu P1
(23)  8031 757140                 mov       blinks,#01000000b        ;miga pozycja numeru wyjscia
(24)  8034 75F007                 mov       B,#7                     ;na pozycji 7 displeja
(25)  8037 120379                 lcall     GETDIGIT                 ;pobierz numer pinu P1: 0..7
(26)  803A 5407                   anl       A,#7                     ;dla bezpieczenstwa wazne tylko 3 lsb
(27)  803C F5F0                   mov       B,A                      ;i przechowanie numeru wyjscia
(28)  803E 757180                 mov       blinks,#10000000b        ;miga znacznik zalaczenia

                                  ;czekanie na rozpoczecie odliczania
(29)  8041 1202C5       wait:     lcall     CONIN                    ;czekanie na klawisz
(30)  8044 B40D57                 cjne      A,#klaw_OK,nieok         ;start - klawisz OK

                        ;te instrukcje beda wykonane gdy wcisnieto klawisz startu odliczania
(31)  8047 1280A4                 lcall     WAITSEK                  ;czekanie sekunde
(32)  804A 757A40                 mov       DL3,#_minus
(33)  804D 757104                 mov       blinks,#00000100b        ;miga kreska miedzy (mm-ss)

(34)  8050 9080CC                 mov       DPTR,#tab_wyj            ;pobierz adres tabeli wyjsc
(35)  8053 E5F0                   mov       A,B                      ;odtworzenie numeru wyjscia
(36)  8055 93                     movc      A,@A+DPTR                ;pobranie maski wyjsc z tabeli
```

```
(37)  8056 F590                    mov    P1,A                 ;i zapisanie do portu P1 - zalaczenie
(38)  8058 757F5C                  mov    DL8,#_o              ;znaczek "o" na DL8 - ON

(39)  805B               ;tutaj odliczanie ustawionego czasu do zera
(40)  805B AB50                    mov    R3,minuty
(41)  805D AC51                    mov    R4,sekundy           ;przechowanie ustawionego czasu

(42)  805F EC            nsek:     mov    A,R4                 ;czy sekundy = 0 ?
(43)  8060 7012                    jnz    niezer
(44)  8062 EB                      mov    A,R3
(45)  8063 601E                    jz     koniec               ;czy minuty = 0 ?
(46)  8065 1280B3                  lcall  DECACC               ;nie, to zmniejszenie minut
(47)  8068 FB                      mov    R3,A
(48)  8069 75F001                  mov    B,#1
(49)  806C 12024E                  lcall  A2HEX                ;wypisanie minut
(50)  806F 7459                    mov    A,#59h
(51)  8071 FC                      mov    R4,A                 ;i korekcja sekund
(52)  8072 8004                    sjmp   wypisz
(53)  8074 1280B3        niezer:   lcall  DECACC               ;zmniejszenie sekund
(54)  8077 FC                      mov    R4,A
(55)  8078 75F004        wypisz:   mov    B,#4
(56)  807B 12024E                  lcall  A2HEX                ;wypisanie sekund
(57)  807E 1280A4                  lcall  WAITSEK
(58)  8081 80DC                    sjmp   nsek                 ;i nastepna sekunda

                         ;te instrukcje gdy zakonczono proces odliczania
(59)  8083 7590FF        koniec:   mov    P1,#255              ;wylaczenie urz¹dzenia
(60)  8086 757100                  mov    blinks,#0            ;wylaczenie migania displeja
(61)  8089 757F71                  mov    DL8,#_F              ;znaczek "-" na DL8 - OFF
(62)  808C 75F001                  mov    B,#1
(63)  808F E550                    mov    A,minuty             ;pokazanie wprowadzonego czasu

(64)  8091 12024E                  lcall  A2HEX                ;minuty
(65)  8094 75F004                  mov    B,#4
(66)  8097 E551                    mov    A,sekundy            ;i sekundy
(67)  8099 12024E                  lcall  A2HEX
(68)  809C 80A3                    sjmp   wait                 ;i skok do momentu czekania na klawisz

(69)  809E B430A0         nieok:   cjne   A,#'0',wait          ;wrowadz czas jeszcze raz - klawisz '0'
(70)  80A1 028000                  ljmp   pocz                 ;i skok na poczatek programu

                         ;------------------------------------------------------------------------------------

(71)  80A4               WAITSEK:                              ;procedura czekania 1 sek.
(72)  80A4 C0E0                    push   Acc
(73)  80A6 E570           czek1:   mov    A,bsw                ;pobranie slowa stanu monitora
(74)  80A8 20E0FB                  jb     Acc.0,czek1          ;czekanie na wyzerowanie bitu 2Hz
(75)  80AB E570           czek2:   mov    A,bsw
(76)  80AD 30E0FB                  jnb    Acc.0,czek2          ;a potem na jego ustawienie
(77)  80B0 D0E0                    pop    Acc
(78)  80B2 22                      ret
                                   ;------------------------------------------------------------------------------------
(79)  80B3               DECACC:                               ;dekrementacja Acc z korekcja dziesietna
(80)  80B3 7003                    jnz    niero
(81)  80B5 7499                    mov    A,#99h
(82)  80B7 22                      ret
(83)  80B8 C3            niero:    clr    C
(84)  80B9 9401                    subb   A,#1
(85)  80BB C0E0                    push   Acc
(86)  80BD 540F                    anl    A,#0Fh
(87)  80BF B40F07                  cjne   A,#0Fh,nie0F
(88)  80C2 D0E0                    pop    Acc
(89)  80C4 54F0                    anl    A,#0F0h
(90)  80C6 4409                    orl    A,#9
(91)  80C8 22                      ret
(92)  80C9 D0E0          nie0F:    pop    Acc
(93)  80CB 22                      ret
                         ;------------------------------------------------------------------------------------
(94)  80CC FEFDFBF7      tab_wyj   db     11111110b,11111101b,11111011b,11110111b
                                          ;wyj: 0,1,2,3
(95)  80D0 EFDFBF7F                db     11101111b,11011111b,10111111b,01111111b
                                          ;wyj: 4,5,6,7
                         ;------------------------------------------------------------------------------------
(96)  80D4                         END
```