Mikrokontrolery? To takie proste...

Część 9
Asembler – język maszynowy procesora

Niniejszy, trzeci z kolei odcinek kursu programowania poświęcony liście instrukcji procesora 8051, jest ostatnim „teoretycznym" kawałkiem niezbędnych informacji, dzięki którym wspólnie krok po kroku utworzymy pierwszy prawdziwy program.
Przy okazji pragnę wspomnieć że w listach napływających od Was drodzy Czytelnicy, często poruszacie sprawę niedosytu wiedzy oraz informacji na temat przytaczanych w lekcjach przykładów. Otóż sprawa kompleksowego a jednocześnie przystępnego przedstawienia problemów związanych z programowaniem mikrokontrolerów nie jest taka prosta, nawet z mego punktu widzenia. Przekonałem się że nie da się w jednym odcinku naszego cyklu przedstawić wszystkich zagadnień użytych w jednej z naszych lekcji. Zbyt wiele informacji „zazębia" się za siebie, toteż proszę o cierpliwość, wszystkie niejasne instrukcje z przykładów zostaną w kolejnym odcinku wyjaśnione do końca.

Jak wspominałem w poprzednim odcinku, przyszła pora na zapoznanie się z ostatnią grupą instrukcji, a mianowicie skokami oraz wywołaniami podprogramów.

Pierwsze z nich dzielimy na:
– skoki warunkowe
– skoki bezwarunkowe

Dodatkowo, w przypadku skoków warunkowych, warunek przy którym następuje skok, może spełniać określony bajt z wewnętrznej pamięci danych procesora (np. kiedy dany bajt ≠ konkretna liczba to następuje skok) lub bit. W tym ostatnim przypadku procesor sprawdza czy bit jest ustawiony (=1) czy wyzerowany (=0) i podejmuje decyzję o skoku lub nie. A co może w programie zmienić takim czy inny bit?, a no jakaś instrukcja która została wykonana wcześniej, a wyniku której dany bit został ustawiony lub wyzerowany. W języku programistów bit (bity), którego znaczenie jest istotne dla działania programu, który informują o tym czy, np. przy dodawaniu dwóch liczb 8–bitowych nastąpiło przeniesienie (kiedy wynik > 255), nazywa się „**flagą**". Określeniem tym będziemy się dość często posługiwać, toteż warto je zapamiętać.

Warto też sobie wyjaśnić co oznacza samo pojęcie skoku fizycznie dla pracy procesora. Otóż jeżeli procesor wykonuje kolejne instrukcje programu, to za każdym pobraniem kolejnej instrukcji zwiększa się automatycznie licznik rozkazów PC, o którym mówiliśmy już wcześniej. Jeżeli procesor napotka na instrukcję skoku, to argumentem (tej instrukcji) w takim przypadku, jest liczba będąca:
– przesunięciem (liczba w kodzie U2, zakres <–128...127>, która jest dodawana do bieżącej wartości PC
  lub
– wartością bezwzględną licznika rozkazów PC (w przypadku skoków bezwarunkowych)

Argument taki zostaje w efekcie dodany do licznika rozkazów PC, co w efekcie powoduje skok do innej części programu – w przód lub w tył.

Na początek poznajmy więc skoki warunkowe testujące bity znajdujące się w obszarze wew. RAM procesora, oczywiście.

JC rel
– ang."jump if Carry" – skocz jeżeli znacznik przeniesienia C jest ustawiony
– sprawdzany jest bit przeniesienia C w słowie PSW (adres bajtu: D0h), jeżeli jest ustawiony (=1) to do licznika rozkazów PC jest dodawana 8–bitowa liczba „rel" (zapisana w kodzie U2), zostaje wykonany skok względny
  PC <– PC + 2, jeśli C=1, to PC <– PC + rel
– kod: 0 1 0 0 0 0 0 0 40h
– cykle: 2 bajty: 2 (kod instrukcji 40h + przesunięcie „rel")
– przykład:

```
    MOV   A,#20h    ;załadowanie do
                    akumulatora liczby 32
    ADD   A,B       ;dodanie do
                    akumulatora bieżącej
                    wartości rej. B
    JC    przenies  ;czy nastąpiło
                    przeniesienie, tak to skocz do
                    etykiety
          ;"przenies"
    MOV   B,#0      ;nie, to wyzeruj rejestr B
    .....           ;po czym wykonuj dalsze
                    instrukcje
    .....
    .....
przenies:           ;tu nastapi skok jeżeli
                    znacznik C był ustawiony
    CLR   C         ;w efekcie nastąpi jego
                    wyzerowanie
```

Jak widać z przykładu korzystając z instrukcji JC nie podawaliśmy argumentu bezpośredniego, czyli liczby, tylko określiliśmy za pomoca etykiety „przenies" miejsce w programie źródłowym do którego ma nastąpić skok, jeżeli warunek będzie spełniony. Taki sposób zapisu jest jak widać prostszy, nie musimy liczyć przesunięcia sami, które to jest zresztą ilością bajtów kodu programu pomiędzy instrukcją skoku JC a miejscem skoku. Ta możliwość dotyczy jednak tych z Was drodzy Czytelnicy, którzy posiadają komputery i mogą skorzystać z kompilatora zawartego na dyskietce AVT–2250/D lub każdego innego.
W przypadku osób które „ręcznie" tłumaczą kod programu, sprawa się nieco komplikuje, ale nie do tego stopnia, aby nie dać sobie z nią rady. W kolejnych lekcjach pokażę jak prosto obliczać przesunięcia w tej instrukcji oraz pozostałych, tam gdzie występuje symbol „**rel**", pamiętajcie o nim przy lekturze dalszej części artykułu!
W tym miejscu istotna uwaga dla wszystkich. Otóż jak widać wartość przesunięcia nie może przekroczyć liczb z zakresu U2 czyli –128...127, innymi słowy skok względny może odbyć się tylko w pewnym „otoczeniu" ("w górę" lub „w dół") od instrukcji skoku. Jeżeli wartość przesunięcia jest ujemna, to skok nastąpi oczywiście w kierunku mniejszych adresów (do PC zostaje dodana liczba ujemna), w przeciwnym przypadku w kierunku adresów wzrastających. W przypadku umieszczenia etykiety (miejsca) skoku poza tym zakresem, prawidłowe obliczenie przesunięcia będzie niemożliwe (dla „ręczniaków"), a w przypadku korzystania z kompilatora 8051 ("komputerowcy") wystąpi komunikat o błędzie kompilacji mówiący o przekroczeniu zakresu skoku względnego. Programiście w takim przypad-

ku nie pozostaje nic innego jak poprawić błąd modyfikując program źródłowy.

JNC rel

– ang."jump if not Carry" – skocz jeżeli znacznik przeniesienia C jest wyzerowany
– sprawdzany jest bit przeniesienia C w słowie PSW (adres bajtu: D0h), jeżeli jest wyzerowany (=0) to do licznika rozkazów PC jest dodawana 8–bitowa liczba „rel" (zapisana w kodzie U2), zostaje wykonany skok względny
PC <– PC + 2, jeśli C=0, to PC <– PC + rel
– kod: 0 1 0 1 0 0 0 0 50h
– cykle: 2 bajty: 2 (kod instrukcji 50h + przesunięcie „rel")
– przykład:

```
    MOV   A,#20h    ;załadowanie do
                    akumulatora liczby 32
dodaj:
    ADD   A,#1      ;dodanie do akumulatora
                    liczby 1
    JNC   dodaj     ;czy nastąpiło przeniesie
                    nie, nie to dodaj jeszcze raz
    .....           ;w efekcie akumulator będzie
                    inkrementowany aż do
    .....           ;czasu kiedy osiągnie
                    wartość 00h, a do
                    znacznika C
                    ;zostanie wpisana „1"–ka,
                    potem zostaną wykonane
    .....           ;dalsze instrukcje
```

JB bit, rel

– ang."jump if Bit Set" – skocz jeżeli bit jest ustawiony
– sprawdzany jest bit, którego adres podany jest w „bit", jeżeli jest ustawiony (=1) to do licznika rozkazów PC jest dodawana 8–bitowa liczba „rel" (zapisana w kodzie U2), zostaje wykonany skok względny
PC <– PC + 3, jeśli (bit)=1, to PC <– PC + rel
– kod: 0 0 1 0 0 0 0 0 20h
– cykle: 2 bajty: 3 (kod instrukcji 20h + adres bitu + przesunięcie „rel")
– przykład:

```
    JB    7Fh, zeruj ;jeżeli bit o adresie 7Fh = 1
                     to skocz do etykiety „zeruj"
    MOV   A,B        ;jeśli nie to dodaj do
                     akumulatora rejestr B
    .....            ;i wykonuj dalsze instrukcje
    .....
zeruj:
    CLR   A          ;jeżeli bit był ustawiony to
                     wyzeruj akumulator
```

JNB bit, rel

– ang."jump if Bit not Set" – skocz jeżeli bit jest wyzerowany
– sprawdzany jest bit, którego adres podany jest w „bit", jeżeli jest wyzerowany (=0) to do licznika rozkazów PC jest dodawana 8–bitowa liczba „rel" (zapisana w kodzie U2), zostaje wykonany skok względny
PC <– PC + 3, jeśli C=0, to PC <– PC + rel
– kod: 0 0 1 1 0 0 0 0 30h
– cykle: 2 bajty: 3 (kod instrukcji 30h + bajt + przesunięcie „rel")
– przykład:

```
    CLR   C          ;wyzerowanie znacznika
                     przeniesienia
zeruj:
    MOV   70h, C     ;wyzerowanie bitu
                     a adresie 70h
    JNB   70h,zeruj  ;jeżeli bit 70h jest =0 to
                     skok do etykiety „zeruj"
```

Podany przykład, z praktycznego punktu widzenia jest nieprzydatny, lecz pokazuje jak można zrealizować pętlę nieskończoną („zawiesić" procesor), używając instrukcji skoku warunkowego. Zauważmy przecież, że warunek: że bit 70h jest wyzerowany będzie spełniony zawsze, toteż za każdym razem nastąpi skok do etykiety „zeruj" w miejsce programu gdzie następuje zerowanie bitu 70h, i cała pętla się powtarza.

JBC bit, rel

– ang."jump if Bit Set and Clear Bit " – skocz jeżeli bit jest ustawiony i wyzeruj go
– sprawdzany jest bit, którego adres podany jest w „bit", jeżeli jest ustawiony (=1) to zostaje wyzerowany po czym do licznika rozkazów PC jest dodawana 8–bitowa liczba „rel" (zapisana w kodzie U2), zostaje wykonany skok względny
PC <– PC + 3, jeśli (bit)=1, to (bit)<–0 i PC <– PC + rel
– kod: 0 0 0 1 0 0 0 0 10h
– cykle: 2 bajty: 3 (kod instrukcji 10h + adres bitu + przesunięcie „rel")
– przykład: wykonanie sekwencji instrukcji

```
    ADD   A, B       ;dodaj do akumulatora
                     zawartość rejestru B
    JBC   C, et01    ;jeżeli nastąpiło przeniesienie
                     to wyzeruj C i skocz
    .....            ;jeżeli nie to wykonuj
                     dalsze instrukcje
    .....
et01:
    MOV   P1, A      ;tu nastąpi skok jeżeli było
                     przeniesienie
```

funkcjonalnie jest równoważne sekwencji:

```
    ADD   A, B       ;dodaj do akumulatora
                     zawartość rejestru B
    JC    et01       ;jeżeli nastąpiło
                     przeniesienie skocz
    .....            ;jeżeli nie to wykonuj
                     dalsze instrukcje
    ......
et01:
    CLR   C          ;tu nastąpi skok jeżeli było
                     przeniesienie, zeruj C
    MOV   P1,A
```

przeanalizuj i zastanów się dlaczego?

To tyle jeżeli chodzi o skoki warunkowe operujące na bitach, teraz pora na kilka instrukcji skoków warunkowych operujących na bajtach, bezwzględnych oraz wywołań podprogramów.

ACALL adr11

– ang. „subroutine call on page", skocz do podprogramu na stronie
– licznik rozkazów PC jest zwiększany o 2, następnie ładowany jest na stos (najpierw młodszy bajt, potem starszy), wskaźnik stosu jest zwiększany o 2, a do bitów 0...10 licznika PC zostaje wpisany 11–bitowy adres bezpośredni podany jako argument instrukcji: „adr11". Pięć najstarszych bitów licznika rozkazów PC nie ulega zmianie, toteż skok zostaje wykonany w obrębie 2 kilobajtowej „strony" pamięci programu, na której występuje następna po „ACALL adr11" instrukcja.
PC <– PC + 2,
SP <– SP + 1, (SP) <– $PC_{7–0}$,
SP <– SP + 1, (SP) <– $PC_{15–8}$
$PC_{10–0}$ <– adr11
– kod: a10 a9 a8 1 0 0 0 1
gdzie: a10, a9, a8 zależą od numeru strony
stąd: 11h, 31h, 51h, 71h, 91h, B1h, D1h, F1h
– cykle: 2 bajty: 2 (kod instrukcji + młodszy bajt adresu „adr11" – bity a7...a0)
– przykład:

```
    ACALL PISZ       ;wywołanie podprogramu
    ......           ;po zakończeniu go
                     zostają wykonane kolejne
    ......           ;instrukcje programu
    ......
```

;tu zaczyna się podprogram, którego (w tym przykładzie) zadaniem jest wysłanie
;poprzez końcówki portu P1 procesora sekwencji kodów ASCII odpowiadających
;kolejnym literom tekstu „Hello from AVT–2250", aż do momentu napotkania kodu
;zerowego ("0").

```
PISZ:               ;etykieta – nazwa
                    podprogramu
    MOV   DPTR, #tekst  ;załaduj do wskaź
                        nika DPTR
                        adres tesktu
next:
    CLR   A         ;wyzeruj akumulator
    MOVC  A,@A+DPTR ;pobierz kolejny
                    znak z tekstu
                    (do akumulatora)
    JZ    koniec    ;czy koniec tekstu ( znak
                    = Acc = 0 )?, tak to skocz
    MOV   P1, A     ;nie to wyślij kod na koń-
                    cówki portu P1
    INC   DPTR      ;przesuń adres na kolejną
                    literę tekstu
    SJMP  next      ;i skocz do „next" aby
                    pobrać kolejną literę
koniec:
    RET             ;instrukcja zakończenia
                    podprogramu i powrotu do
                    ;głównej części programu
    ;a to jest zdefiniowany przykładowy tekst,
    zakończony kodem (bajtem) „0"
    ;w celu identyfikacji końca tekstu.
tekst   DB'   HELLO FROM AVT–2250, 0
```

LCALL adr16

– ang. „subroutine call" – skocz do podprogramu
– licznik rozkazów PC zostaje zwiększony o 3, jest ładowany na stos w efekcie czego wskaźnik stosu jest zwiększany o 2, a do licznika rozkazów PC zostaje wpisany 16–bitowy adres bezpośredni, podany jako argument przy wywołaniu instrukcji „LCALL". Dzięki tej instrukcji możliwe jest wywołanie podprogramu w dowolnym obszarze pamięci (64kB) poprzez podanie bezpośredniego adresu.
PC <– PC + 3
SP <– SP + 1, (SP) <– $PC_{7–0}$,
SP <– SP + 1, (SP) <– $PC_{15–8}$
PC <– adr16
– kod: 0 0 0 1 0 0 1 0 12h
– cykle: 2 bajty: 3 (kod instrukcji 12h + bardziej znaczący bajt „adr16" + mniej znaczący bajt „adr16"
– przykład: wypisanie liczby 12h na wyświetlaczu komputerka edukacyjnego przy pomocy instrukcji A2HEX standardowo umieszczonej w kodzie monitora systemu (w EPROM dołączonej do zestawu).

```
A2HEX   EQU   024Eh   ;deklaracja
                      adresu podpro-
                      gramu w
                      monitorze
        MOV   A, #12h ;wypisz liczbę
                      12h jako 2 znaki
                      w HEX
        MOV   B, #3   ;na pozycji
                      3 wyświetlacza
        LCALL A2HEX   ;pod adresem
                      024Eh znajduje
                      się gotowy
        ;podprogram do wypisywania
        akumulatora w
        ;postaci 2 cyfr HEX
```

W efekcie wykonania przykładu na wyświetlaczach zostanie wypisana liczba 12h. W postaci „12".

RET
- ang. „return from subroutine", powróć z podprogramu
- adres powrotu z podprogramu (zapamiętana wcześniej podczas instrukcji ACALL lub LCALL wartość licznika PC) jest wpisywany do licznika rozkazów PC. Wskaźnik stosu zostaje zmniejszony o 2. W efekcie następuje powrót z podprogramu. Instrukcją tą musi kończyć się każdy podprogram (z wyjątkiem podprogramów obsługi przerwań, wywoływanych automatycznie przez procesor).
  PC_{15-8} <– (SP), SP <– SP – 1
  PC_{7-0}, <– (SP), SP <– SP – 1
- kod: 0 0 1 0 0 0 1 0 22h
- cykle: 2 bajty: 1
- przykład: patrz procedura „PISZ" przy okazji omawiania instrukcji ACALL. Procedura „PISZ" została zakończona instrukcją powrotu z podprogramu „RET", dzięki której mogło nastąpić odtworzenie pierwotnego stanu licznika PC i dzięki temu powrót do programu głównego. Tak powinien kończyć się każdy Twój podprogram.

RETI
- ang. „return from interrupt", powróć z podprogramu obsługi przerwania
- adres powrotu z podprogramu obsługi przerwania jest wpisywany do licznika rozkazów PC. Wskaźnik stosu zostaje mniejszony o 2. Instrukcją tą musi kończyć się każdy podprogram, który jest wywoływany automatycznie przez procesor w przypadku zgłoszenia przerwania i żądania jego obsługi.
  PC_{15-8} <– (SP), SP <– SP – 1
  PC_{7-0}, <– (SP), SP <– SP – 1
- kod: 0 0 1 1 0 0 1 0 32h
- cykle: 2 bajty: 1
- przykład: omówimy przy okazji dokładnego opisu systemu przerwań oraz możliwości praktycznych zastosowań.

Teraz zapoznam Was, drodzy Czytelnicy z trzema bardzo często używanymi instrukcjami skoków bezwarunkowych. W liście instrukcji procesora 8051 istnieją trzy rodzaje instrukcji skoku bezwarunkowego. Róznica między nimi polega na „odległości" (o ile bajtów programu dalej można skoczyć) pomiędzy instrukcją skoku a miejscem w które następuje skok. I tak mamy do czynienia ze skokiem :
- krótkim (instrukcja SJMP – „short jump"), gdzie skoczyć możemy w obrębie 256 bajtów (w góre o 127 lub w dół o 128) od miejsca wywołania instrukcji SJMP. W tym przypadku argumentem instrukcji jest przesunięcie (tak jak wspominany wczesniej parametr „rel" przy okazji omawiania insturkcji skoków warunkowych operujących na bitach), wyrażone poprzez liczbę 8–bitowa zapisaną w kodzie U2.
- na stronie (instrukcja AJMP – „absolute jump on page"), gdzie skok może nastapić w obszarze 2 kilobajtowej strony. W tym przypakdu argumentem instukcji jest 11–bitowy adres bezpośredni, który podajemy po mnemoniku AJMP. W praktyce można ten skok określić jako „średni" (w stosunku do krótkiego SJMP)
- długi (instrukcja LJMP – „long jump"), gdzie skok może nastapić w dowolne miejsce w obszarze 64kB przestrzeni pamięci programu procesora. W tym przypadku argumentem skoku jest 16–bitowy adres bezpośredni podawany jako argument tej instrukcji.

Ktoś może w tym miejscu zapytać, „po co ta komplikacja, przecież wystarczyłby w zasadzie tylko jeden typ skoku, taki który daje największe możliwości, czyli LJMP?". Tak ale w przypadku pisania programów na procesory jednoukładowe istnieje potwierdzona w praktyce zasada, że: „każdy bajt kodu jest cenny, toteż program należy pisać tak aby zajmował jak najmniej miejsca". I tu leży sedno sprawy, otóż trzy wymienione rodzaje instrukcji skoków różnią sie objętością zajmowaną w kodzie programu. I tak skok krótki SJMP oraz na stronie AJMP zajmuja 2 bajty kodu, natomiast skok długi LJMP zajmuje 3 bajty, ktoś powie że to mała różnica, otóz moi drodzy będziecie mogli się o tym jeszcze przekonać że często 1 dodatkowy bajt wolny w programie to recepta na zrealizowanie i dokończenie niejednego programu. Przejdźmy zatem do szczegółow.

AJMP adr11
- ang. „Absolute Jump", skocz bezwarunkowo na stronie
- licznik rozkazów zostaje zwiększony o 2, następnie do jego bitów 0–10 zostaje wpisany 11–bitowy adres bezpośredni podany jako parametr. Pozostałe 5 bardziej znaczących bitów nie zmienia się. W efekcie wykonania tego polecenia następuje skok pod adres na stronie pamięci programu o wielkości 2 kilobajty, na której jest umieszczona kolejna instrukcja po AJMP.
  PC_{10-0} <– adr11
- kod: a10 a9 a8 0 0 0 0 1 ,gdzie: a10, a9, a8 zależą od numeru strony stąd: 01h, 21h, 41h, 61h, 81h, A1h, C1h, E1h
- cykle: 2 bajty: 2 (kod instrukcji + młodszy bajt adresu „adr11" – bity a7...a0)
- przykład:

```
        MOV     P1,A
        AJMP    koniec
        .....
        .....
        .....
koniec:
```

Po przesłaniu zawartości akumulatora do portu P1 (na końcówki tego portu) procesor ominie instrukcje w liniach oznaczonych wielokropkiem i wykona skok do miejsca (etykiety) programu oznaczonej jako „koniec".

SJMP rel
- ang. „Shot Jump" – skok bezwarunkowy krótki
- do zawartości licznika rozkazów PC jest dodawane 8–bitowe przesunięcie (liczba ze znakiem w kodzie U2 z zakresu <–128...127>). W efekcie wykonywany jest skok w obrębie 256 bajtów od kolejnej po SJMP instrukcji.
  PC <– PC + 2, PC <– PC + rel
- kod: 1 0 0 0 0 0 0 0 80h
- cykle: 2 bajty: 2 (kod instrukcji 80h + przesunięcie „rel")
- przykład:

```
    MOV   A,B       ;dowolne insturkcje
                    programu
    CLR   B         ;———————— „" ————————
    .....
stop:
    SJMP  stop      ;i przykład pętli
                    nieskończonej, którą
                    powinien dla
                    ;bezpieczeństwa kończyć
                    się każdy program
    END             ;deklaracja końca
                    programu (to nie jest
                    instrukcja!)
```

A oto pozostałe instrukcje skoków, jedna bezwarunkowa oraz pozostałe warunkowe sprawdzające warunek równości bajtu o podanym adresie lub rejestru z innym bajtem lub argumentem bezpośrenim.

Dokończenie listy instrukcji w następnym numerze.

Sławomir Surowiński

Lekcja 4

W dzisiejszej lekcji poćwiczymy sobie użycie niektórych z przedstawionych instrukcji wywołań podprogramów. Na wstępie jednak kilka wyjaśnień. Otóż jak zdążyliście się zorientować z poprzednich lekcji w przykładach podawałem listingi programów, w których występowały instrukcje wywołań „dziwnych" procedur np.

LCALL A2HEX (1)
lub
LCALL CLS (2)

Otóż moi drodzy w programie monitora który macie w swoim komputerku w pamięci EPROM zawarty jest program monitora – o tym już wiecie. Dzięki niemu możliwa jest komunikacja i ładowanie programów z komputera PC lub ręcznie. Monitor jest na tyle mądry że dodatkowo komunikuje się z użytkownikiem za pomocą 8–mio pozycyjnego wyświetlacza i lokalnej klawiatury. No ale przecież ten „monitor" to w końcu też kawałek programu napisany za pomocą tych samych instrukcji , które poznawaliście przez ostatnie 3 odcinki naszego kursu. To właśnie dzięki niemu całe urządzenie żyje i pozwala na niezłą zabawę.

Oprócz zawartych w monitorze funkcji uaktywnianych klawiszami np. „LOAD", „EDIT", itp. istnieje kilka bardzo użytecznych procedur które są potrzebne no chociażby do wypisania aktualnej wartości rejestru DPTR na wyświetlaczu (tak się dzieje np., przy edycji pamięci RAM kiedy na 4 pierwszych wyświetlaczach DPTR jest wyświetlany jako 4 znaki w kodzie szesnastkowym.

A czy zastanawiałeś się drogi Czytelniku, jak to się dzieje, że po naciśnięciu przez Ciebie odpowiedniego klawisza, układ reaguje

np. uaktywniając odpowiednią funkcję monitora? Do tego celu wykorzystywany jest inny podprogram (także zawarty w monitorze) którego zadaniem jest oczekiwanie na naciśnięcie klawisza a następnie podjęcie decyzji „co z tym fantem dalej robić...". Takie przykłady można by mnożyć.

Ja w każdym razie w kolejnym odcinku przedstawię Ci pełną listę dodatkowych procedur (podprogramów) które będą niezmiernie przydatne przy tworzeniu wspólnych programów.

Ci z „komputerowców", którzy już nabyli dyskietkę AVT2250/D mogą zapoznać się z taką listą czytając zbiór „BIOS.INC", w którym są zawarte definicje adresów wszystkich potrzebnych procedur . Dodatkowo umieszczono w nich opis i sposób wywołania, czyli parametry wejściowe podprogramu, oraz efekt działania.

Pamiętaj, wszystkie one są częścią programu monitora i tak samo jak napisany przez Ciebie program są ciągiem określonych instrukcji, w wyniku działania których otrzymujesz określony efekt, np. na wyświetlaczu.

Ze względu na ograniczona objętość artykułu, w dzisiejszej lekcji wykorzystamy tylko niektóre procedury (podprogramy).

Dla przykładu skorzystajmy z podprogramu którego zadaniem jest wypisanie na wyświetlaczu na pozycji określonej w rejestrze B, liczby znajdującej się w akumulatorze w postaci heksadecymalnej (szesnastkowej). Procedura ta znajduje się w monitorze pod adresem **024Eh**.

Ciekawa jest też procedura CLS – w wyniku wykonania jej całe pole odczytowe zostaje wyczyszczone. Adres tego podprogramu w monitorze to **0274h**. Procedura ta jest bezparametrowa, wystarczy ją wywołać aby uzyskać zamierzony efekt.

Zadanie

Wypisać dowolną 8–bitową liczbę na wyświetlaczu w postaci heksadecymalnej.

Załóżmy że chcemy aby liczba pojawiła się na 3–ciej i 4–tej pozycji, należy wykonać i skompilować (komputerowo lub ręcznie przetłumaczyć) instrukcje:

```
                CPU     8052.DEF

0045            liczba  equ     45h             ;liczba do wyswietlenia
0003            pozycja equ     3               ;pozycja na ktorej ma byc
                                                ;wyswietlona liczba
8000            org     8000h

8000 120274             lcall   CLS             ;wyczyszczenie wyswietlacza
8003 7445               mov     A, #liczba      ;zaladowanie liczby
8005 75F003             mov     B, #pozycja     ;i pozycji na displeju
8008 12024E             lcall   A2HEX           ;wyswietlenie

800B 80FE       stop:   sjmp    stop            ;stop programu

800D            END
```

Pamiętajmy, że jest to listing programu, czyli że komputerowcy wpisują tylko deklaracje i instrukcje wraz z ewentualnymi komentarzami (po średniku), bez pierwszej kolumny liczb określającej adres bieżącej instrukcji (tekst pochyły), oraz bez drugiej kolumny liczb w której zawarty jest przetłumaczony kod (tekst pogrubiony) niezbędny dla „ręczniaków" – przetłumaczcie sami i porównajcie wynik!

Przyjrzyjmy się listingowi dokładnie i przeanalizujmy go linia po linii. Dla uproszczenia będę używał dodatkowych oznaczenia (K) w przypadku gdy linia ma znaczenie tylko dla „komputerowców", oraz (R) gdy linia ma znaczenie dla „ręczniaków". gdy nie występuje żadne z oznaczeń, informacja jest istotna dla wszystkich. A więc zaczynamy (Lx – numer linii, np. L4 – linia nr 4)

L1: (K), deklaracja zbioru który zawiera niezbędne definicje rejestrów procesora 8052 (a także 8051 oczywiście)

L2: deklaracja EQU – równoważności, oznacza że w dalszej części programu słowo „liczba" jest równoważne (literowo) wyrażeniu „45h", a 45h to zapisana w kodzie szesnastkowym liczba.

L3: deklaracja EQU – podobnie jak poprzednio, tym razem słowo „pozycja" będzie oznaczało wyrażenie „3", u nas jest to numer pozycji na wyświetlaczu, na której ma być wyświetlona 2–pozycyjna liczba z akumulatora.

L4: deklaracja ORG – nakazuje kompilatorowi (K) przetłumaczenie (kompilację) instrukcji występujących po tej deklaracji począwszy od adresu wskazanego w parametrze ORG, czyli w naszym przypadku będzie to 8000h – początek zewnętrznej pamięci w komputerku edukacyjnym. Dla „ręczniaków" jest to informacja aby wprowadzać kod programu (pogrubiony tekst) od adresu 8000h korzystając oczywiście z funkcji „EDIT" komputerka.

Stąd zaczyna się prawdziwa część programu.

L5: dobrze by było aby po rozpoczęciu wykonywania naszego programu wyświetlacz nie był zapisany jakimiś znakami, zupełnie nam niepotrzebnymi. W tym celu profilaktyczne wywołujemy podprogram CLS – czyszczący wyświetlacz.

L6: do akumulatora zostaje załadowana liczba do wyświetlenia, tym razem jest to liczba 45h. Te polecenie można zapisać także jako: „ mov A, #45h ".

„Ręczniacy" tak będą musieli postąpić, bowiem nie korzystają z mądrego kompilatora, który wie że od momentu deklaracji w linii 2 (L2) słowo „liczba" oznacza „45h". Popatrzcie zresztą na kod tej linii po lewej stronie (pogrubione liczby) – 7445 = 74 45, pierwsza to kod instrukcji „mov A, #dana "(patrz wkładka z tabelą instrukcji 8051), druga to argument, czyli wspomniana liczba 45h – prawda że się zgadza!

L7: musimy także powiedzieć procedurze A2HEX, żeby wyświetliła liczbę od pozycji 3 na displeju, toteż ładujemy jako drugi parametr „pozycję" (która oznacza po prostu „3")..

Uff!

L8: wreszcie właściwe wywołanie podprogramu A2HEX. „Ręczniacy" będą musieli wpisać adres bezpośredni procedury, który podałem wcześniej, stąd w linii po lewej stronie listingu mamy sekwencję kodu: 12 **02 4E**, pierwsza liczba to kod instrukcji LCALL, następne dwie to adres procedury A2HEX – sprawdź!

W efekcie po wykonaniu tego podprogramu na wyświetlaczach DL3 i DL4 pojawi się liczba „45", czyli to co chcieliśmy!

L8: tu mamy omawiany przy okazji prezentowania instrukcji skoków bezwarunkowych, przykład instrukcji SJMP , która została użyta do zatrzymania programu. Instrukcja użyta w ten sposób spowoduje że procesor będzie skakał w kółko pod adres etykiety „stop", czyli pod adres pod którym znajduje się instrukcja SJMP ... „i tak w koło Macieju...", w efekcie można powiedzieć że program będzie krążył w pętli nieskończonej, i tylko klawisz „M" – powrotu do monitora może nas wybawić z tego ślepego zaułka.

Ktoś może powiedzieć, „.... ale po co właściwie ta instrukcja SJMP, przecież i tak to już koniec programu...".

Koniec dla nas drodzy Czytelnicy, my o tym doskonale wiemy, bo takie było założenie, ale biedny procesor, kiedy wróci z podprogramu A2HEX, będzie „pożerał" kolejne bajty kodu programu, i jeżeli nie napotka na sensowne zakończenie w pętli nieskończonej (jak w przykładzie) zacznie pobierać kolejne bajty programy spoza adresu 800Dh, a tam co jest?, a no same „śmieci" Kochani – sprawdźcie to funkcją EDIT monitora.

Nie musze tłumaczyć, co się wtedy może stać. Nielogiczna sekwencja bajtów spowoduje niekontrolowane działanie procesora, w efekcie czego najprawdopodobniej układ dojdzie do końca pamięci programu FFFFh po czym licznik rozkazów PC zostanie wyzerowany a w związku z tym na wyświetlaczu zobaczymy znajomy napis „HELLO", oznajmiający nam że właśnie zrestartowaliśmy nasz komputerek. Dodatkowo stanie się to tak szybko że efekt naszej pracy zostanie nie zauważony przez nasze czujne oko.

Dlatego stosowanie na końcu programu takiego skoku jest wskazane w każdym przypadku.

No dobrze skoro wiecie o co chodzi, to radzę poeksperymentować z innymi wartościami akumulatora oraz zmieniać np. pozycję wyświetlania, pamiętając że może ona zawierać się w granicach 1...7.

W kolejnej lekcji omówię wszystkie pozostałe procedury dostępne w monitorze oraz podam ich adresy wywołań, wtedy będziemy modli naprawdę „poszaleć". Nabywcy dyskietek AVT2250/D mogą się z nimi już teraz zapoznać czytając zbiory BIOS.INC i CONST.INC.

Wesołej zabawy!

Sławomir Surowiński

Też to potrafisz

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 nastapi 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 ety
                            kiety „jest_zero"
nie_zero:                   ;jeżeli nie to wykonuj po
                            zostałe instrukcje
    .....
    .....
jest_zero:                  ;tu nastapi 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 po
                            zostałe 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 wczesniej. W efekcie zostaje wykonany skok w programie. Tu uwaga, skok nastepuje względem instrukcji występującej po instrukcji CJNE. Dodatkwo 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 zalezności 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 wciśniętego 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 kody
                              wcisniętego
                              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, jeżlei 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 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 liczba 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 róznią 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 występuje 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 zalezności 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

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

jest równowazna 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.

```
    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 sie stan procesora, z wyjątkiem licznika rozkazów którym po pobraniu tej instrukcji jest zwiekszany 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
  będzie zajmie procesorowi 1 cykl maszynowy więcej niz sekwencja
  MOV A, B
  MUL A, B
  ale efekt działania będzie taki sam w obu przypadkach.

LEKCJA 4

W dzisiejszej lekcji poćwiczymy sobie użycie instrukcji wywołań podprogramów do realizacji prostych układów licznikowych. Na wstepnie jednak kilka wyjasnień. Otóż jak zdążyliście sie zorientowac z poprzednich lekcji w przykładach podawałem listingi programów, w których występowały instrukcje wywołań „dziwnych" procedur np.

LCALL A2HEX (1)

oraz operacje przesłania „niezrozumiałych" argumentów bezpośrednich np.:

MOV DL1, #_minus (2)

Otóż moi drodzy w programie monitora który macie w swoim komputerku w pamięci EPROM zawarty jest program monitora – o tym juz wiecie. Dzięki niemu możliwa jest komunikacja i ładowanie programów z komputera PC lub ręcznie. Monitor jest na tyle mądry że dodatkowo komunikuje sie z użytkownikiem za pomoca 8–mio pozycyjnego wyświetlacza i lokalnej klawiatury. No ale przecież ten „monitor" to w końcu też kawałek programu napisany za pomocą tych samych instrukcji, które poznawaliście przez ostatnie 3 odcinki naszego kursu. To własnie dziękiniemu całe urządzenie żyje i pozwala na niezłą zabawę.

Oprócz zawartych w monitorze funkcji uaktywnianych klawiszami np. „LOAD", „EDIT", itp. istnieje kilka bardzo użytecznych procedur które sa potrzeben ano chociażby do wypisania aktualnej wartości rejestru DPTR na wyświetlaczu (tak się dzieje np., przy edycji pamięci RAM kiedy na 4 pierwszych wyświetlaczach DPTR jest wyświetlany jako 4 znaki w kodzie szesnastkowym.

A czy zastanawiałeś sie drogi Czytelniku, jak to sie dzieje, że po nacisnięciu przez Ciebie odpowiedniego klawisza, układ reaguje np. uaktywniając odpowiednią funkcję monitora? Do tego celu wykorzystywany jest inny podprogram (także zawarty w monitorze) którego zadaniem jest oczekiwanie na nacisnięcie klawisza a nastepnie podjęcie decyzji „co z tym fantem dalej robić...". Takie przykłady można by mnozyć.

Ja w każdym razie w kolejnym odcinku przedstawię Ci pełną listę dodatkowych procedur (podprogramów) które beda niezmiernie przydatne przy tworzeniu wspólnyhc programów.

Ci z „komputerowców", którzy juz nabyli dyskietkę AVT2250/D mogą zapoznać się z taką listą czytając zbiór „BIOS.INC", w którym są zawarte definicje adresów wszysktich portrzebnych procedur . Dodatkowo umieszczono w nich opis i sposób wywołania, czyli parametry wejściowe porprogramu, oraz efekt działania.

Pamiętaj, wszystkie one są częścią programu monitora i tak samo jak napisany przez Ciebie program są ciągiem określonych instrukcji, w wyniku działania których otrzymujesz określony efekt, np. na wyświetlaczu.

Ze względu na ograniczona objetość artykułu, w dzisiejszej lekcji wykorzystamy tylko niektóre procedury (podprogramy).

Dla przykładu skorzystajmy z podprogramu którego zadaniem jest wypisanie na wyświetlaczu na pozycji określonej w rejestrze B, liczby znajdującej się w akumulatorze w postaci heskadecymalnej (szesnastkowej). Procedura ta znajduje się w monitorze pod adresem 024Eh.

Zadanie 1

Wypisać dowolną 8–bitową liczbę na wyświetlaczu w postaci heksadecymalnej.

Załóżmy że chcemy aby liczba pojawiła się na 3–ciej i 4–tej pozycji, należy wykonać i skompilować (komputerowo lub ręcznie przetłumaczyć) instrukcje:

```
        MOV     A, #liczba
;jako liczba wpisac dowolna wartość np. 45h
        MOV     B, #3       ;na DL3 i DL4
        LCALL   A2HEX
```

Sławomir Surowiński