W kolejnym odcinku naszego cyklu o mikrokontrolerach 8051 kontynuujemy szczegółowy opis instrukcji procesora ze zwróceniem uwagi na składnię instrukcji oraz skrótowe przykłady zastosowania. Podczas analizy z pewnością przyda się zamieszczona we wkładce poprzedniego numeru EdW, tabela ze spisem wszystkich instrukcji. I choć dzisiejszy odcinek może wydać się nieco monotonny, to trzeba pamiętać, że nauka każdego języka, tak porozumiewania się jak i programowania inteligentnych układów elektronicznych jest niezbędna każdemu elektronikowi - hobbyście.

Kontynuując opis instrukcji przedstawimy kolejno grupę najważniejszych instrukcji dotyczących przemieszczania danych w strukturze rejestrów procesora. Dodatkowe instrukcje umożliwiające skoki warunkowe oraz wywoływanie podprogramów, przedstawimy w ostatniej, trzeciej części opisu asemblera.

Operacje przemieszczenia danych

1. Instrukcja **MOV**

Instrukcją służącą do przekazywania danych pomiędzy rejestrami procesora pamięcią wewnętrzną jest: „MOV" (ang. „move" – przesuń, przenieś)

W zależności od tego co i gdzie „przenosimy", polecenie to może mieć kilkanaście różnych postaci w zależności od zastosowanych argumentów. Jednocześnie warto wiedzieć, że polecenie to w praktyce nie powoduje dosłownego „przemieszczenia danej, lub zawartości rejestru, ale jej skopiowanie ze źródła do miejsca przeznaczenia.

Ogólne instrukcję MOV można zapisać jako:

MOV <d> <s>

gdzie: <d> jest miejscem przeznaczenia –tam dokąd ma być skopiowana dana ze źródła (ang. „destination") , a <s> źródłem pobrania danej (ang. „source"). W wyniku wykonania instrukcji MOV zawartość źródła <s> zostaje umieszczona (skopiowana) w obiekcie przeznaczenia <d>. We wszystkich przypadkach (oprócz jednego) argumentami instrukcji MOV są wyrażenia 8–bitowe: rejestry, dane adresy pośrednie itp. Jedynie załadowanie 16–bitowego wskaźnika adresu DPTR wymaga odpowiedniego 16–bitowego (stałej) argumentu.

Poniżej opiszemy wszystkie możliwe przypadki użycia instrukcji MOV.

MOV A, Rn

– do akumulatora zostaje załadowana zawartość rejestru Rn
  A <– Rn
– kod: 1 1 1 0 1 n2 n1 n0 gdzie n2...n0 wskazują na R0...7 stąd: E8h–EFh
– cykle: 1 bajty: 1
– przykład:
```
MOV  R7,#0
MOV  A, R7   ;wyzerowanie akumulatora
.....
```

MOV A, adres

– do akumulatora zostaje załadowana zawartość komórki wewnętrznej pamięci RAM o adresie: „adres"
  A <– (adres)
– kod: 1 1 1 0 0 1 0 1 E5h
– cykle: 1 bajty: 2 (kod instrukcji E5h + 8–bitowy adres)
– przykład:
```
MOV  A, 20h   ;załadowanie do A
              zawartości komórki
              o adresie 20h
```

MOV A, @Ri

– do akumulatora zostaje załadowana zawartość komórki wewn. RAM, której adres znajduje się w rejestrze R0 (i=0), lub R1 (i=1)
  A <– (Ri) gdzie i = 0 lub 1
– kod: 1 1 1 0 0 1 1 i stąd: E6h, E7h
– cykle: 1 bajty: 1
– przykład:
```
                ;przed wywołaniem
                instrukcji R1 zawiera
                wartość 34h
MOV  A, @R1     ;załadowanie zawartości
                komórki o adresie 34h do
                ;akumulatora
```

MOV A, #dana

– instrukcja załadowania 8–bitowej liczby „dana" do akumulatora
  A <– dana
  kod: 0 1 1 1 0 1 0 0 74h
– cykle: 1 bajty: 2 (kod instrukcji: 74h + dana)
– przykład:
```
MOV  A,#100   ;załadowanie do
              akumulatora liczby 100
```

MOV Rn, A

– do rejestru Rn (R0...R7) zostaje załadowana zawartość akumulatora
  Rn <– A
– kod: 1 1 1 1 1 n2 n1 n0 gdzie n2...n0 wskazują na R0...7 stąd: F8h–FFh
– cykle: 1 bajty: 1
– przykład:
```
MOV  R3, A    ;przepisanie zawartości
              akumulatora do rejestru R3
```

MOV Rn, adres

– do rejestru Rn (R0...R7) zostaje załadowana zawartość komórki o adresie „adres"
  Rn <– (adres) gdzie n = 0...7
– kod: 1 0 1 0 1 n2 n1 n0 gdzie n2...n0 wskazują na R0...7 stąd: A8h–AFh
– cykle: 1 bajty: 2 (kod instrukcji + adres)
– przykład:

```
MOV   R4, 65h   ;załadowanie do rejestru
                R4 zawartości komórki
                o adresie 65h
```

MOV Rn, #dana
- do rejestru Rn (R0...R7) zostaje wpisana 8–bitowa liczba
  Rn <– dana gdzie n = 0...7
- kod: 0 1 1 1 1 n2 n1 n0 gdzie n2...n0 wskazują na R0...7 stąd: 78h–7Fh
- cykle: 1 bajty: 2 (kod instrukcji + dana)
- przykład: jeżeli chcemy np. wpisać do rejestru R6 liczbę 16 można wydać komendę

```
MOV   R6, #10h  ;10h szesnastkowo to 16
                dziesiętnie
```

MOV adres, A
- do komórki o adresie „adres" zostaje wpisana zawartość akumulatora
  (adres) <– A
- kod: 1 1 1 1 0 1 0 1 F5h
- cykle: 1 bajty: 2 (kod instrukcji + adres)
- przykład:

```
MOV   00h, A    ;wpisanie zawartości
                akumulatora do komórki
                o adresie 0
                ;instrukcja równoznaczna
                zapisowi MOV R0, A w
                ;przypadku gdy
                aktywnym zbiorem rej.
                roboczych jest 0.
```

MOV adres, Rn
- do komórki o adresie „adres" zostaje wpisana zawartość rejestru Rn (R0...R7)
  (adres) <– Rn gdzie n = 0...7
- kod: 1 0 0 0 1 n2 n1 n0 gdzie n2...n0 – wskazują na R0...7 stąd: 88h–8Fh
- cykle: 2 bajty: 2 (kod instrukcji + adres)
- przykład:

```
MOV  30h, R5    ;wpisanie do komórki
                w wew. RAM zawartości
                rejestru  ;R5.
```

MOV adres1, adres2
- przepisanie zawartości komórki o adresie „adres2" do komórki o adresie „adres1"
  (adres1) <– (adres2)
- kod: 1 0 0 0 0 1 0 1
- cykle: 2 bajty: 3 (kod instrukcji + adres2 + adres1)
- przykład:

```
MOV   7Fh, 7Eh  ;przepisanie zawartości
                dwóch sąsiadujących
                komórek w
                ;wew. RAM procesora
```

MOV adres, @Ri
- do komórki o adresie „adres" zostaje wpisana zawartość komórki której adres znajduje się w rejestrze R0 (i=0) lub R1 (i=1)
  (adres) <– (Ri)
- kod: 1 0 0 0 0 1 1 i gdzie i = 0 lub 1 stąd: 86h, 87h
- cykle: 2 bajty: 2 (kod instrukcji + adres)
- przykład:

```
MOV   R1, #32
MOV   32h, @R1  ;przepisanie zawartości
                komórki o adresie 32h na nia
                ;samą (fizycznie operacja
                ta nie ma efektu)
```

MOV adres, #dana
- do komórki o adresie „adres" zostaje wpisana wartość stałą (8–bitowa liczba)
  (adres) <– stała
- kod: 0 1 1 1 0 1 0 1 75h
- cykle: 2 bajty: 3 (kod instrukcji + adres + dana)
- przykład:

```
MOV   45h, #100;komórce o adresie 45h
zostaje nadana wartość 100
```

MOV @Ri, A
- do komórki o adresie znajdującym się w rejestrze R0 (i=0) lub R1 (i=1) zostaje wpisana zawartość akumulatora
  (Ri) <– A
- kod: 1 1 1 1 0 1 1 i gdzie i = 0 lub 1 stąd: F6h, F7h
- cykle: 1 bajty: 1
- przykład:

```
CLR   A         ;wyzerowanie akumulatora
MOV   @R1, A    ;wyzerowanie komórki
                o adresie w R1
```

MOV @Ri, adres
- do komórki o adresie znajdującym się w rejestrze R0 (i=0) lub R1 (i=1) zostaje wpisana zawartość komórki o adresie „adres"
  (Ri) <– (adres)
- kod: 1 0 1 0 0 1 1 i gdzie i = 0 lub 1 stąd: A6h, A7h
- cykle: 2 bajty: 2 (kod instrukcji + adres)
- przykład:

```
MOV   R0, #11h
MOV   @R0,10h   ;przepisanie zawartości
                komórki o adresie 10h do
                komórki
                ;sąsiedniej o adresie 11h
```

MOV @Ri, #dana
- do komórki o adresie znajdującym się w rejestrze R0 (i=0) lub R1 (i=1) zostaje wpisana wartość stała (liczba)
  (Ri) <– dana
- kod: 0 1 1 1 0 1 1 i gdzie i = 0 lub 1 stąd: 76h, 77h
- cykle: 1 bajty: 2 (kod instrukcji + dana)
- przykład:

```
MOV   @R0, #255 ;jeżeli wcześniej
                rejestr R0 miał
                wartość np. 30h, to w
                ;efekcie tej operacji
                do komórki o adresie
                30h zostanie
                ; wpisana liczba 255
```

MOV DPTR, #dana16
- instrukcja załadowania 16–bitowego, bezwzględnego adresu do wskaźnika danych DPTR
  DPTR <– dana16 gdzie „dana16" jest liczbą 16–bitową (zakres: 0...FFFFh)
- kod: 1 0 0 1 0 0 0 0 90h
- cykle: 2 bajty: 3 (kod instrukcji + starszy bajt + młodszy bajt liczby „dana16")
- przykład:

```
MOV DPTR,#0     ;wyzerowanie wskaźnika
                danych
```

2. Instrukcja **MOVC**

Podobną do instrukcji MOV jest MOVC. Służy ona także do przemieszczania danych z tym że przemieszczanie dotyczy tylko pobierania danych (bajtów) znajdujących się w kodzie programu, czyli w wewn. lub zewnętrznej pamieci programu procesora. W praktyce instrukcję tę wykorzystuje się do pobierania danych stałych (np. tablic przy konwersji arytmetycznej lub logicznej. Innym często spotykanym przypadkiem jest generowanie standardowych komunikatów (tekstowych) np. na wyświetlaczach LCD w określonych sytuacjach pracy procesora w celu poinformowania użytkownika o konkretnym zdarzeniu. Ponieważ takie komunikaty są z reguły niezmienne, w praktyce programista umieszcza je w kodzie programu (pamięci stałej).

Dzięki takiemu działaniu instrukcja MOVC umożliwia odczytanie całego kodu programu użytkownika, co często w praktyce nie jest pożądane, bo pozwala na np. „nielegalny" odczyt i skopiowanie przez osobę niepowołaną („hackera") programu utworzonego przez programistę. Jest to oczywiście naruszeniem praw autorskich danego projektu, ale łamiącemu prawo piratowi pozwala np. na powielenie ciekawego urządzenia (np. sterownika) bez zgody jego autora. Na szczęście procesor 8051 i wszystkie z jego rodziny mają wbudowane mechanizmy sprzętowego zabezpieczenia przed taką sytuacją. Istnieje bowiem możliwość permanentnego zablokowania instrukcji MOVC wywoływanej z zewnętrznej pamięci programu (napisanego np. przez hackera). Sytuacja ta dotyczy oczywiście aplikacji wykorzystujących procesory z wewnętrzną pamięcią programu (87xx, 89xx) gdzie przezorny programista umieścił kod programu w wewnętrznej pamięci programu, uniemożliwiając tym skopiowanie go przez osoby niepowołane. W przypadku aplikacji z kodem programu umieszczonego w zewnętrznej pamięci programu (np. EPROM) nie istnieje możliwość zabezpieczenia programu – hacker może w po prostu wyjąć z układu pamięć EPORM (ROM) i odczytać ją na dowolnym programatorze (bez stosowania instrukcji MOVC) .

MOVC A, @A+DPTR
- do akumulatora zostaje załadowana dana z pamięci programu spod adresu będącego sumą bieżącej wartości wskaźnika danych DPTR i zawartości akumulatora. Najpierw procesor tworzy 16–bitowy adres poprzez dodanie DPTR i A potem pobiera spod tego adresu daną (bajt kodu programu) i umieszcza ją z akumulatorze.
  A <– (A + DPTR)
- kod: 1 0 0 1 0 0 1 1 93h
- cykle: 2 bajty: 1
- przykład:

```
MOV     DPTR, #tablica  ;załadowanie
                        adresu tablicy
                        do wskaźnika
                        DPTR
MOV     A,#3            ;pobierz czwarty
                        element tablicy
                        (nie trzeci, bo od
                        ;elementy są
                        numerowane
                        od zera)
MOVC  A, @A+DPTR        ;pobranie
                        elementu –
                        litera 'A'
.....                   ;gdzie tablica
                        może być
                        zdefiniowana
                        w programie
                        jako np.:
tablica DB      ' WITAJ KOLEGO!'
```

MOVC A, @A+PC
- do akumulatora zostaje załadowana dana z pamięci programu spod adresu będącego sumą wartości: licznika rozkazów PC (na-

stępnej po tej instrukcji) i zawartości akumulatora. W praktyce przy wykonaniu tej instrukcji, adres pobrania jest równy sumie zawartości akumulatora oraz wartości licznika rozkazów – będzie to adres następnej po MOVC instrukcji.
A <– (A + PC)
– kod: 1 0 0 0 0 0 1 1 83h
– cykle: 2 bajty: 1
– przykład:

```
CLR     A    ;wyzerowanie akumulatora
MOVC  A, @A+PC  ;pobranie do
                akumulatora kodu
                rozkazu NOP (00h)
NOP
.....
```

3. Instrukcja **MOVX**

Instrukcja MOVX służy do przesyłania danej pomiędzy akumulatorem a zewnętrzną pamięcią danych. Wykonanie tej instrukcji uaktywnia sygnały /RD (przy odczycie z zewnętrznej pamięci) lub /WR (przy zapisie) procesora – piny P3.7 i P3.6. Dodatkowo porty P0 i P2 pełnią wtedy rolę magistrali systemowej dzięki której wystawiany jest adres oraz przekazywana dana do zewnętrznej pamięci danych.

Jak już zapewne wiesz z poprzednich odcinków naszego cyklu zewnętrzna pamięć danych można zaadresować w dwojaki sposób.

Pierwszą metodą jest użycie pełnego 16–bitowego adresu. W takim przypadku procesor odczytując lub zapisując daną w tej pamięci (właśnie dzięki instrukcji MOVX) młodszą część adresu zatrzaskuje w zewnętrznym latch u (np. 74573), starszą wystawia na port P2.

Często jednak używana kostka pamięci SRAM jest mniejszej pojemności i większość linii adresowych starszego bajta (adresu) nie jest wykorzystana do sterowania pamięcią. W takim przypadku możliwe jest adresowanie pamięci za pomocą tzw. „stronicowania". Ten sposób omówiliśmy już w poprzednich odcinkach szkoły mikroprocesorowej. Jak zapewne pamiętasz w takim trybie adresowania procesor przy obsłudze zewnętrznej pamięci danych wystawia tylko młodszą część adresu (A0...A7), natomiast port P2 nie jest modyfikowany, co pozwala użytkownikowi na pełna kontrolę sposobu i kierunku ustawienia jego końcówek – a więc jest metodą na maksymalne wykorzystanie cechy „jednoukładowości" procesora.

I tak dwa wspomniane dwa tryby adresowania zewnętrznej RAM maja swoje odzwierciedlenie w liście instrukcji z wykorzystaniem rozkazu MOVX, oto one.

Tryb pełnego adresu (16–bitowego)

MOVX A, @DPTR

– do akumulatora zostaje załadowana dana (bajt) z zewnętrznej pamięci danych (odczyt z zewnętrznej pamięci danych) spod adresu w DPTR
A <– (DPTR)
– kod: 1 1 1 0 0 0 0 0 E0h
– cykle: 2 bajty: 1
– przykład:
aby odczytać zawartość komórki z zewn. RAM o adresie np. 1240h nalezy wykonać instrukcje:

```
MOV   DPTR, #1240h ;załadowanie
                   adresu (16–bit) do
                   wskaźnika DPTR
MOVX  A, @A+DPTR   ;i odczyt danej spod
                   tego adresu
.....              ;dana znajduje się
                   w akumulatorze
```

MOVX @DPTR, A

– do komórki zewnętrznej pamięci danych o podanym w DPTR adresie zostaje przesłana zawartość akumulatora – innymi słowy jest to operacja zapisu do zewnętrznej pamięci danych.
(DPTR) <– A
– kod: 1 1 1 1 0 0 0 0 F0h
– cykle: 2 bajty: 1
– przykład:
aby zapisać daną w obszarze zewnętrznej pamięci danych pod adresem np. 8000h należy wykonać instrukcje:

```
MOV    A, .....  ;w miejsce kropek należy
                 wpisać źródło danej
MOV    DPTR, #8000h  ;podajemy też
                     adres zapisu
MOVX   @DPTR,A       ;i zapisujemy
                     daną w zewn.
                     pamięci
```

Tryb stronicowania (niepełnego adresu)

MOVX A, @Ri

– do akumulatora zostaje przesłana zawartość komórki w obszarze zewn. pamięci danych spod adresu znajdującego się w rejestrze R0 (i=0) lub R1 (i=1): adres 8–bitowy
A <– (Ri) gdzie Ri = R0, lub R1
– kod: 1 1 1 0 0 0 1 i gdzie i = 0,1
stąd: E2, E3
– cykle: 2 bajty: 1
– przykład:
niech w układzie z procesorem 8951 pracującym z wewn. pamięcią programu znajduje się zewnętrzna pamięć danych w postaci kostki SRAM 2kB – typ 6116.
Linie adresowe A0...A7 tej pamięci są dołączone do zatrzasku młodszej części adresu szyny procesora (patrz poprzednie odcinku cyklu). Trzy starsze linie A8...A10 są dołączone np. do pinów P2.0, P2.1 i P2.2 procesora, pozostałe końcówki portu P2 (P2.2...P2.7) są wykorzystywane np. do sterowania przekaźnikami jakiegoś urządzenia zewnętrznego. Aby odczytać daną z tej pamięci np. spod adresu 24h na stronie pierwszej (strony liczone od 0 do 7, bo 2kB / 256 = 8 stron) należy wykonać instrukcje:

```
CLR    P2.2    ;wyzerowanie linii
               adresowej A10
CLR    P2.1    ;wyzerowanie linii
               adresowej A9
SETB   P2.0    ;ustawienie linii
               adresowej A8 (strona: 1)
MOV    R1, #24h    ;załadowanie
                   adresu komórki
                   do wskaźnika
MOVX   A, @R1  ;i przesłanie jej zawartości
               do akumulatora
.....
```

Zauważ że przy takim zaadresowaniu pamięci nie uległy modyfikacji piny P2.3...P2.7 portu P2 procesora, co w wielu przypadkach jest wręcz niezbędne. Można by oczywiście zaadresować tę pamięć za pomocą instrukcji MOVX A, @DPTR (podając wtedy adres MOV DPTR, #0124h), ale wtedy zniszczeniu uległy by stany pozostałych, nie dołączonych do pamięci, końcówek portu P2.

MOVX @Ri, A

– do obszaru zewnętrznej pamięci danych o adresie znajdującym się w rejestrze Ri (R0 gdy i=0 lub R1 gdy i=1) zostaje przesłana zawartość akumulatora. Innymi słowy jest to zapis do zewnętrznej pamięci danych.
(Ri) <– A gdzie Ri = R0, lub R1
– kod: 1 1 1 1 0 0 1 i gdzie i = 0,1
stąd: F2, F3
– cykle: 2 bajty: 1
– przykład:
weźmy sytuacje z poprzedniego przykładu, ale tym razem chcemy zapisać daną znajdującą się w akumulatorze do komórki o adresie 00h na stronie 7 (innymi słowy fizyczny adres komórki będzie równy: 0700h), w tym celu należy wykonać ciąg instrukcji:

```
SETB   P2.2    ;ustawienie linii
               adresowej A10
SETB   P2.1    ;ustawienie linii
               adresowej A9
SETB   P2.0    ;ustawienie linii
               adresowej A8 (strona: 7)
MOV    R1, #00h    ;załadowanie
                   adresu komórki
                   do wskaźnika
MOVX   @R1, A  ;i zapisane danej
               z akumulatora
               w zewn. RAM
.....
```

4. Instrukcje przesyłania wymiany danych ze stosem

PUSH adres

– ang. „push onto stack", prześlij na stos
– w wyniku tej operacji zawartość wskaźnika stosu SP jest zwiększana o 1, po czym na wierzchołek stosu (adresie w wew. RAM wskazywanym przez SP) zostaje zapisana zawartość komórki z wew. RAM o podanym adresie bezpośrednim „adres". Innymi słowy wykonywana jest operacja „przesłania na stos".
SP <– SP + 1, (SP) <– (adres)
– kod: 1 1 0 0 0 0 0 0 C0h
– cykle: 2 bajty: 2 (kod instrukcji C0h + adres)
– przykłady:

```
PUSH    ACC    ;przesłanie akumulatora
               na stos
PUSH    B      ;przesłanie rejestru B
               na stos
PUSH    20h    ;przesłanie zawartości
               komórki o adresie 20h
               na stos
PUSH    DPH    ;przesłanie 16–bitowego
               wskaźnika danych na stos
PUSH    DPL    ;w dwóch instrukcjach,
               starsza i młodsza część
               DPTR
.....
```

POP adres

– ang. „pop from stack" , zdejmij ze stosu
– w wyniku tej operacji dana znajdująca się pod adresem w wewn. RAM określonym w SP zostaje wpisana do komórki o podanym adresie bezpośrednim „adres". Następnie wskaźnik stosu SP zostaje zmniejszony o 1. Innymi słowy wykonywana jest operacja „zdjęcia ze stosu".

(adres) <– (SP), SP <– SP – 1
– kod: 1 1 0 1 0 0 0 0 D0h
– cykle: 2 bajty: 2 (kod instrukcji D0h + adres)
– przykład: jeżeli w poprzednim przykładzie załadowaliśmy na stos rejestry w w/w kolejności, to aby poprawnie je odtworzyć należy zdjąć je w kolejności odwrotnej, czyli:

```
POP   DPL
POP   DPH     ;odtworzenie wskaźnika
              DPTR
POP   20h     ;następnie komórki
              o adresie 20h
POP   B       ;i rejestru B
POP   ACC     ;wreszcie akumulatora
....
```

5. Dodatkowe instrukcje przemieszczania danych

XCH A, Rn

– ang. „exchange register with accumulator", wymień akumulator z rejestrem
– w wyniku tej operacji zawartość akumulatora zostaje wymieniona z zawartością rejestru Rn (R0...R7), wymieniona tzn. że liczba znajdująca się w A znajdzie się w Rn, i odwrotnie.
A <–> Rn
– kod: 1 1 0 0 1 n2 n1 n0 gdzie n2...n0 wskazują na R0...7 stąd: C8h–CFh
– cykle: 1 bajty: 1
– przykład:

```
MOV   A, #10    ;nadanie wartości
                akumulatorowi
MOV   R4, #23   ;nadanie wartości
                rejestrowi R4
XCH   A, R4     ;wymiana danych
....            ;teraz w A jest liczba 23,
                a w R4 liczba 10
```

XCH A, adres

– w wyniku tej operacji zawartość akumulatora zostaje wymieniona z zawartością komórki w wew. pamięci RAM o podanym adresie bezpośrednim
A <–> (adres)
– kod: 1 1 0 0 0 1 0 1 C5h
– cykle: 1 bajty: 2 (kod instrukcji C5 + adres)
– przykład:

```
MOV   A,#99
MOV   00h, #0
XCH   A, 00h    ;teraz w akumulatorze
                będzie 0
....
```

XCH A, @Ri

– w wyniku tej operacji zawartość akumulatora zostaje wymieniona z zawartością komórki w wew. RAM o adresie znajdującym się w rejestrze R0 (i=0) lub R1 (i=1)
A <–> (Ri)
– kod: 1 1 0 0 0 1 1 i gdzie i = 0,1 stąd: C6h, C7h
– cykle: 1 bajty: 1
– przykład: wykonanie instrukcji

```
MOV   R1, #20h
CLR   A
XCH   A, @R1
```

jest równoważne

```
MOV   A, 20h
MOV   20h, #0
```

zastanów się dlaczego ?

XCHD A, @Ri

– w wyniku tej operacji młodszy półbajt (bity 3–0) akumulatora zostaje wymieniony z młodszym półbajtem komórki w wew. RAM o adresie zawartym w rejestrze Ro (i=0) lub R1 (i=1).
$A_{3–0}$ <–> $(Ri)_{3–0}$
– kod: 1 1 0 1 0 1 1 i gdzie i = 0,1 stąd: D6h, D7h
– cykle: 1 bajty: 1
– przykład:

```
MOV   A, #5Ah
MOV   @R1, #A5h
XCHD  A, @R1
....            ;teraz w akumulatorze
                będzie liczba 55h, a pod
                ;adresem @R1 liczba AAh
```

Operacje na bitach

Do tej pory omawialiśmy instrukcje operujące na bajtach danych. Procesor 8051 i mu pochodne zawiera bardzo pomocny zestaw instrukcji do wykonywania operacji na pojedynczych bitach. Dzięki temu możliwe jest wykonanie wielu często niezbędnych operacji, jedna z nich było omówione wcześniej stronicowanie zewnętrznej pamięci danych (przykład z instrukcją MOV @Ri,A). Trzeba wiedzieć że większość rejestrów specjalnych SFR procesora posiada możliwość bezpośredniego adresowania ich bitów. I tak np. akumulator (A) składa się z 8–miu adresowanych bitów Acc.7 ... Acc.0

Toteż aby np. ustawić wybrane bity tego rejestru nie trzeba modyfikować całości a jedynie wyzerować lub ustawić wybrani bit. Dla przykładu prześledzimy sytuację kiedy chcemy wyzerować bit 4 akumulatora bez ingerowania w pozostałe, można wykonać te zadanie dwojako:
– poprzez instrukcję iloczynu logicznego:

```
ANL   A, #11101111b ;wyzerowanie
                    bitu 4
                    akumulatora
```

– lub poprzez instrukcję działającą na pojedynczym bicie:

```
CLR   Acc.4
```

Instrukcje operujące na bitach nabierają szczególnie praktycznego znaczenia przy badaniu stanu końcówek (portów) mikroprocesora lub przy ich sterowaniu (ustawianiu na nich poziomów logicznych niskich lub wysokich oraz przy ustawianiu w stan wysokiej impedancji). Problemem tym zajmiemy się przy okazji kolejnej lekcji ze szkoły mikroprocesorowej.

CLR C

a) ang. „clear carry", zeruj flagę przeniesienia
b) w wyniku tej operacji wyzerowany zostaje znacznik (bit w rejestrze PSW) przeniesienia C
C <– 0
c) kod: 1 1 0 0 0 0 1 1 C3h
d) cykle: 1 bajty: 1
e) przykład:

```
CLR   C       ;wyzerowanie przeniesienia
SUBB  A, B    ;aby odjąć A – B bez pożyczki
....
```

SETB C

a) ang. „set carry", ustaw flagę przeniesienia
b) w wyniku tej operacji ustawiona zostaje flaga przeniesienia
C <– 1
c) kod: 1 1 0 1 0 0 1 1 D3h
d) cykle: 1 bajty: 1
e) przykład:

```
SETB  C
ADDC  A,#0    ;inkrementacja
              akumulatora
              z wykorzystaniem C
....
```

CLR bit

a) ang. „clear bit" , zeruj bit
b) w wyniku tej operacji wyzerowany zostaje bit którego adres podany jest bezpośrednio (bit) <– 0
c) kod: 1 1 0 0 0 0 1 0 C2h
d) cykle: 1 bajty: 2 (instrukcja C2 + adres bitu)
e) przykład:

```
CLR   AFh     ;zablokowanie systemu
              przerwań (EA w słowie
              IE = 0)
....
```

Przykład ten można także zapisać w postaci

```
CLR   EA
```

SETB bit

a) ang. „set bit", ustaw bit
b) w wyniku tej operacji ustawiony zostaje bit którego adres podany jest bezpośrednio (bit) <– 1
c) kod: 1 1 0 1 0 0 1 0 D2h
d) cykle: 1 bajty: 2 (instrukcja + adres bitu)
e) przykład:

```
SETB  P1.2    ;ustawienie wysokiego
              poziomu logicznego
              ;na wyprowadzeniu
              2 portu P1 (pin 3 procesora)
```

CPL C

a) ang. „complement carry" , zaneguj flagę przeniesienia
b) w wyniku tej operacji flaga C zostaje zanegowana
C <– /C
c) kod: 1 0 1 1 0 0 1 1 B3h
d) cykle: 1 bajty: 1
e) przykład:

```
SETB  C       ; C=1
CPL   C       ;teraz C=0
```

CPL bit

a) ang. „complement bit" , zaneguj bit
b) w wyniku tej operacji zanegowany zostanie bit, którego adres podany jest bezpośrednio (bit) <– /(bit)
c) kod: 1 0 1 1 0 0 1 0 B2h
d) cykle: 1 bajty: 1
e) przykład: wykonanie instrukcji

```
neg:  CPL   P1.0
      SJMP  neg
```

spowoduje nieprzerwane generowanie na pinie 1 procesora fali prostokątnej o częstotliwości równej: ..... (jako ćwiczenie powinieneś sam wpisać wartość – odpowiedź w kolejnym odcinku naszego cyklu).

ANL C, bit

a) ang. „AND direct bit to carry" , iloczyn logiczny znacznika C i bitu o adresie podanym jako bezpośredni
b) w wyniku tej operacji zostanie wykonany iloczyn logiczny flagi przeniesienia C oraz bitu o adresie „bit", a wynik zostanie umieszczony w C
C <– C ∩ (bit)
c) kod: 1 0 0 0 0 0 1 0 82h
d) cykle: 2 bajty: 2 (kod instrukcji + adres bitu)
e) przykład:

```
ANL   C, D6h  ;iloczyn flagi przeniesienia
              C i flagi przeniesienia
              ;pomocniczego AC
              (jej adres w słowie
              PSW to D6h)
```

Też to potrafisz

ANL C, /bit

a) ang. „AND complement of direct bit to carry", iloczyn logiczny znacznika C i zanegowanego bitu o adresie podanym jako bezpośredni
b) w wyniku tej operacji zostanie wykonany iloczyn logiczny flagi przeniesienia C oraz zanegowanego bitu o adresie „bit", a wynik zostanie umieszczony w C
C <– C ∩ /(bit)
c) kod: 1 0 1 1 0 0 0 0 B0h
d) cykle: 2 bajty: 2 (kod instrukcji + adres bitu)
e) przykład:

```
ANL   C, /D7h   ;równoważne
                wyzerowaniu znacznika C
                ;bo wymnożono C przed
                zanegowane C
                ;(adres C w PSW to D7h)
```

ORL C, bit

a) ang. „OR direct bit to carry", logiczna suma znacznika C i bitu o adresie podanym jako bezpośredni
b) w wyniku tej operacji zostanie wykonana suma logiczna flagi przeniesienia C oraz bitu o adresie „bit", a wynik zostanie umieszczony w C
C <– C (bit)
c) kod: 0 1 1 1 0 0 1 0 72h
d) cykle: 2 bajty: 2 (kod instrukcji + adres bitu)
e) przykład:

```
ORL   C, D6h    ;suma flagi przeniesienia
                C i flagi przeniesienia
                ;pomocniczego AC
                (jej adres w słowie PSW
                to D6h)
```

ORL C, /bit

a) ang. „OR complement of direct bit to carry", suma logiczna znacznika C i zanegowanego bitu o adresie podanym jako bezpośredni
b) w wyniku tej operacji zostanie wykonana suma logiczna flagi przeniesienia C oraz zanegowanego bitu o adresie „bit", a wynik zostanie umieszczony w C
C <– C ∪ /(bit)
c) kod: 1 0 1 0 0 0 0 0 A0h
d) cykle: 2 bajty: 2 (kod instrukcji + adres bitu)
e) przykład:

```
ORL   C, /D7h   ;równoważne ustawieniu
                znacznika C
                ;bo zsumowano C przed
                zanegowane C
                ;(adres C w PSW to D7h)
```

MOV C, bit

a) ang. „move direct bit to carry" , przenieś zawartość bitu o podanym adresie bezpośrednim do znacznika przeniesienia C
b) w wyniku tej operacji zawartość bitu o podanym adresie „bit" zostanie przepisana do znacznika przeniesienia C
C <– (bit)
c) kod: 1 0 1 0 0 0 1 0 A2h
d) cykle: 1 bajty: 2 (kod instrukcji A2h + adres bitu)
e) przykład:

```
MOV   C, P1.7   ;odczyt stanu końcówki
                7 portu P1 i wpisanie go
                do C
```

MOV bit, C

a) ang. „move carry to direct bit", przenieś zawartość znacznika C do bitu o podanym adresie bezpośrednim
b) w wyniku tej operacji zawartość znacznika przeniesienia C zostanie przepisana do bitu o podanym adresie bezpośrednim „bit"
(bit) <– C
c) kod: 1 0 0 1 0 0 1 0 92h
d) cykle: 2 bajty: 2 (kod instrukcji + adres bitu)
e) przykład: – pytanie?...
jak przenieść zawartość jakiegoś bitu (o podanym adresie bezpośrednim do innego o innym adresie bezpośrednim – czyli wykonać operację :
(bit2) <– (bit1)
Otóż do wykonania tego niezbędny jest znacznik C, nie można bowiem wykonać takiej operacji bezpośrednio jednym rozkazem. Nie istnieje w 8051 instrukcja która by wykonywała taką operację przeniesienia zawartości jednego bitu do drugiego bezpośrednio.

Załóżmy że bit1 zdefiniowaliśmy jako:

```
bit1  EQU  20h  ;EQU to deklaracja
                równoważności
                – przypisania
                ;nazwie umownej
                (wyrazowi) konkretnego
                adresu
```

a bit 2 jako

```
bit2  EQU  30h
```

Chcąc teraz przenieść zawartość bitu 1 do bitu 2 często początkujący wykonują błędną operację:
MOV bit2, bit1
co w efekcie nie powoduje przeniesienia bitu 1 do 2 ale przepisanie zawartości komórki wew. RAM o adresie 20h do komórki o adresie 30h – wykonywana jest zatem operacja na bajtach , a nie przenoszenie bitów (patrz instrukcja: *MOV adres1, adres2*)
Prawidłową odpowiedzią na postawione zadanie będzie wykorzystania instrukcji MOV działających na bitach i znaczniku przeniesienia C w postaci sekwencji:

```
MOV   C, bit1   ;najpierw załaduj bit1 do C
MOV   bit2, C   ;a potem C do bitu 2,
                w efekcie przeniosłeś bit1
                do 2
```

....

W dzisiejszym odcinku to wszystkie instrukcje, w trzeciej części – ostatniej opisującej listę poleceń procesora 8051 zapoznasz się drogi Czytelniku z instrukcjami skoków warunkowych, bezwarunkowych i do podprogramów. Po zapoznaniu się z pozostałymi poleceniami rozpoczniemy wspólne tworzenie pierwszych aplikacji (programów) na procesor 8051 przy wykorzystaniu zmontowanego przez Ciebie komputerka edukacyjnego.

Na razie w celu przećwiczenia omówionych dzisiaj instrukcji zapraszam do wnikliwej lektury „Lekcji 3" szkoły mikroprocesorowej.

Sławomir Surowiński

Lekcja 3

Jako uwieńczenie analizy drugiej części listy instrukcji procesora 8051 proponuję abyśmy przećwiczyli przesyłanie danych pomiędzy procesorem a zewnętrzną pamięcią danych, umieszczoną na płytce komputerka – układ U4. Otóż wspólnie napiszemy prostą procedurę testującą wszystkie komórki tej pamięci, z dodatkowym wyświetlaniem na wyświetlaczu aktualnie testowanego adresu i ewentualnych błędów. Wykorzystamy przy tym m.in. instrukcję MOVX, a pamięć będziemy adresować z wykorzystaniem pełnego 16–bitowego adresu.

Popatrzmy zatem i przeanalizujmy listing przykładowego programu, realizującego procedurę testowania pamięci.

Program zasadniczo składa się z 3 części:

– w pierwszej na wyświetlaczu wypisywany adres aktualnie testowanej komórki
– w drugiej: testowana jest komórka pamięci przez zapisanie w niej, odczyt i weryfikację wartości testowej: czyli dwóch liczb: 55h i AAh. Wybrano właśnie takie liczby, bowiem jak pewnie zauważyłeś są one kombinacjami zer i jedynej na przemian: 55h = 01010101b, a AAh = 10101010h, co pozwala na sprawdzenie wszystkich 8–miu bitów danej komórki pamięci.
– w trzeciej części inkrementowany jest adres wskaźnika DPTR po czym jeżeli nie przekracza on założonego obszaru pamięci (8100h...9FFFh) wykonywany jest skok na początek procedury i testowana jest następna komórka.

W przypadku wykrycia błędu następuje skok do etykiety „blad" w której w efekcie zostaje wypisany komunikat „Err" – skrót od „error" – z angielskiego „błąd" wraz z widniejącym adresem wadliwej komórki.

W przypadku przetestowania całego zakresu pamięci wypisany zostaje komunikat o zakończeniu: „End" – koniec.

Zauważmy że testowany obszar zaczyna się od adresu 8100h, a nie np. 8000h, dlaczego? Otóż pamiętajmy że w obszarze 8000h....8040h znajduje się kod naszego programu, toteż gdybyśmy wpisali jako wartość początkową adres 8000h, program zostałby zamazany i komputerek zawiesiłby się.

Ta sama uwaga dotyczy adresu końcowego, w przypadku pamięci 8kB (6264). Posiadacze komputerków z 32kB RAM (62256) mogą zakończyć testowanie pamięci na adresie FFFFh.

W takim przypadku należy nieco zmodyfikować program, sposób w dalszej części artykułu.

Uwaga, przed rozpoczeciem wpisywanie programu upewnij się czy zwora JP3 znajduje się w pozycji „8000h".

Dla komputerowców:

Przedstawiony program znajduje się na dyskietce kursu (AVT–2250/D) jako zbiór *LEKCJA3.S03*.

Należy go skompilować jak poprzednie lekcje wydając komendę

DO LEKCJA3 lub użyć bezpośrednio komendy kompilatora:

> PASM51 LEKCJA3.S03 /H /L

co spowoduje utworzenie zbioru wynikowego w postaci Intel–HEX, gotowego do przesłania do komputerka poprzez łącze szeregowe.

Dodatkowy zbiór *LEKCJA3.LST* jest listingiem programu przedstawionym w artykule.

W przypadku chęci przetestowania pamięci 32kB (posiadacze U4 – 62256) należy wykasować linię :

```
8024  64A0    xrl  A,#A0h  ;testujemy RAM
```

a następnie przekompilować program źródłowy jeszcze raz. Wtedy w momencie gdy po inkrementacji DPTR będzie = 0000h (przekroczy FFFFh czyli ostatni adres w RAM) warunek :

```
        mov     A, DPH
        jnz     next
```

i tak będzie spełniony bez skasowanej linii i w efekcie nastąpi skok na koniec programu.

Dla nie posiadających PC ta:

Tak jak poprzednio należy wstukać kod programu począwszy od adresu 8000h, korzystając z gotowych bajtów kodu podanych w drugiej kolumnie listingu 1, czyli:

„12, 02, 74, 90, 81, 00 .... itd.

W przypadku chęci przetestowania pamięci 32kB (U4 – 62256) należy pominąć linię z listingu pod adresem 8024h, wpisując w miejsce bajtów :

```
        64, A0
```

bajty

```
        00, 00
```

czyli dwie instrukcje NOP (puste).

Dodatkowo można poeksperymentować z początkową wartością wskaźnika DPTR (inną niż proponowana: 8100h). W tym miejscy uwaga: „pójście wyżej adresu 8041h spowoduje zamazanie części kodu programu testującego – patrz listing 1.

Jeżeli denerwuje Cię szybkie przemiatanie testowanego adresu, możesz to spowolnić wpisując w linii pod adresem 8006h większą wartość do akumulatora.

Reguła: wartość wpisana do A * 2 milisekundy = opóźnienie po teście każdej komórki.

Komputerowcy moga zmodyfikować tę linię w programie LEKCJA3.S03

```
np.     mov     A, #99h
        .....
```

przy takiej wartości procedura testująca będzie trwała niesamowicie długo (można się zdrzemnąć).

Pozostali moga modyfikować opóźnienie poprzez wpisanie do komórki o adresie 8007h (funkcja „Edit" monitora) dowolną wartość z zakresu 1...FFh (1...255) i próbowac od nowa.

Wesołej zabawy !

Sławomir Surowiński

```
CPU 8052.DEF
                ;*****************************************************************
                 ; Lekcja 3: Testowanie zewnetrznej pamieci danych U4
                 ;*****************************************************************
include         const.inc
include         bios.inc                                ;deklaracje dla kompilatora

8000                            org     8000h           ;kod programu od adresu 8000h

8000 120274                     lcall   CLS             ;wyczyszczenie wyswietlacza
8003 908100                     mov     DPTR,#8100h     ;testujemy od adresu 8100h do 9FFFh
8006            next:
8006 7405                       mov     A,#5            ;male opoznienie (ok. 10ms)
8008 120295                     lcall   DELAY           ;abys mogl zaobserwowac zmiane adresu
800B 75F001                     mov     B,#1            ;od pozycji 1 wyswietlacza
800E 12025F                     lcall   DPTR4HEX        ;wypisz aktualnie testowany adres RAM
8011            test55:
8011 7455                       mov     A,#55h          ;zaladowanie 1 liczby testowej (01010101b)
8013 F0                         movx    @DPTR,A         ;i zapisanie jej w zewn. RAM
8014 E0                         movx    A,@DPTR         ;i odczyt spod tego samego adresu

8015 6455                       xrl     A,#55h          ;sprawdzenie danej ze wzorcem
8017 701A                       jnz     blad            ;jezeli sie nie zgadza to blad komorki
8019            testAA:
8019 74AA                       mov     A,#AAh          ;zaladowanie 2 liczby testowej (10101010b)
801B F0                         movx    @DPTR,A         ;i zapisanie jej w zewn. RAM
801C E0                         movx    A,@DPTR         ;i odczyt spod tego samego adresu

801D 64AA                       xrl     A,#AAh          ;sprawdzenie danej ze wzorcem
801F 7012                       jnz     blad            ;jezeli sie nie zgadza to blad komorki

8021 A3                         inc     DPTR            ;zwiekszenie wskaznika adresu o 1
8022 E583                       mov     A,DPH           ;spr. czy nie koniec testowanego adresu
8024 64A0                       xrl     A,#A0h          ;testujemy RAM do adresu A000h
8026 70DE                       jnz     next            ;jezeli nie to testuj nastepna komorka

8028            koniec:
8028 757D79                     mov     DL6,#_E
802B 757E54                     mov     DL7,#_n         ;na koniec milutki napis „End" – koniec
802E 757F5E                     mov     DL8,#_D         ;informujacy ze koniec testu
8031 8009                       sjmp    stop            ;i skok na koniec programu

                ;_____________________________________________________________________
                ;ta czesc programu zadziala gdy bedzie blad jakiejs komorki RAM

8033 757D79     blad:           mov     DL6,#_E
8036 757E50                     mov     DL7,#_r         ;niemmily komunikat ze pod adresem z DPTR
8039 757F50                     mov     DL8,#_r         ;jest blad pamieci RAM – napis „Err"
803C 7571E0     stop:           mov     blinks,#E0h     ;aby napis „End" lub „Err" bedzie mrygal

803F 80FE       stop2:          sjmp    stop2           ;i stop programu (klawisz M – powrot)

8041                            END
```