Sterowanie graficznych wyświetlaczy z telefonów komórkowych firmy Nokia, część 3

W ostatniej części cyklu przedstawiamy procedury umożliwiające wyświetlanie na wyświetlaczu graficznym znaków alfanumerycznych, bargrafów, prostych grafik oraz linii. Z myślą o najbardziej niecierpliwych Czytelnikach przedstawiamy także sposób sterowania wyświetlacza za pomocą komputera PC z zainstalowanym odpowiednim oprogramowaniem.

Pisanie na ekranie

Jeśli mamy już za sobą procedurę inicjalizacji wyświetlacza, możemy przystąpić do wyświetlania informacji. Zaczniemy od wyświetlania tekstów. Jak już wcześniej sygnalizowałem, do tego celu będzie potrzebny tzw. generator znaków. W przypadku mikrokontrolerów będzie to po prostu obszar pamięci ROM (Flash) mikrokontrolera z zapisanymi wzorcami poszczególnych znaków ASCII. Stosując znaki o wielkości 5x7 pikseli, zapisanie jednego znaku będzie wymagało 5 bajtów pamięci. Aby poszczególne znaki były od siebie oddzielone, należy dodać jeszcze jeden - 6 bajt. Zakładając możliwość wyświetlania znaków ASCII o kodach od 0 do 127, potrzebujemy 128*6 = 768 bajtów pamięci Flash. Znaki o kodach poniżej 32 (32 to spacja) mogą być wykorzystane do wyświetlania dodatkowych symboli lub semigrafiki. W tym miejscu wyjaśnię, dlaczego warto stosować generator zawierający 6 bajtów na znak, zamiast 5 bajtów i automatycznego dodawania przerwy po każdym znaku. Otóż możemy tworzyć znaki semigraficzne większe niż wielkość jednego znaku - poprzez składanie ich z kilku znaków. Jeśli zastosujemy metodę automatycznego dodawania przerwy, to wtedy taka semigrafika będzie poprzedzielana przerwami. Jeśli jednak nie zamierzamy wykorzystywać semigrafiki, to zmieniając generator znaków można w ten sposób oszczędzić 128 bajtów pamięci.

Jeśli wyświetlacz został zainicjowany w trybie adresowania poziomego, wyświetlenie jednego znaku na ekranie polega na obliczeniu adresu, pod którym znajduje się jego wzorzec w generatorze znaków, a następnie na pobraniu 6 bajtów z obszaru generatora i wysłaniu ich na wyświetlacz poprzez SPI. To wszystko. Proste, prawda?

No dobrze, ale to tylko jeden znak - powiecie. Nie szkodzi, po wysłaniu tego znaku wyświetlacz jest gotowy do przyjęcia kolejnych 6 bajtów tworzących następny znak tekstu. Jego pozycję będą wyznaczały liczniki wierszy i kolumn wyświetlacza, które zgodnie z **rys. 3**, przy adresowaniu poziomym automatycznie wskażą na kolejną pozycję znaku lub - jeśli dotrzemy do końca linii - automatycznie przeniosą nas do następnej linii. Pozostaje jeszcze jedna przydatna funkcja *gotoXY*, umożliwiająca rozpoczęcie wyświetlania danych w określonym miejscu wyświetlacza. Jej realizacja jest również bardzo prosta, bo wystarczy wymaganą pozycję X znaku pomnożyć przez 6, i komendą *Set X address* wysłać do licznika kolumn, a wymaganą pozycję Y wpisać bezpośrednio do licznika wierszy komendą *Set Y address*. Na **list. 2** przedstawiono przykład owych procedur. Jest to oczywiście jeden z wielu możliwych sposobów wypisywania tekstów na wyświetlaczu - w miarę ubogi w możliwości, lecz jednocześnie jeden z najprostszych i wymagający niewielu zasobów procesora, a w szczególności zapotrzebowania na pamięć RAM. Zawartość przykładowego generatora znaków znajduje się na CD-EP7/2003B - plik ASCII.DEF. Jest on zapisany w formacie pseudodefinicji asemblera używanego przez kompilator AVR-GCC.

List. 2. Przykład procedury umożliwiającej wyświetlanie znaków alfanumerycznych zdefiniowanych w pliku ASCII.DEF na graficznym LCD

```
lcd_data:                   ; wyświetlenie znaku o kodzie w R24
      push   r30            ; zachowaj rejestr Z - potrzebne aby
      push   r31            ; łatwiej wyświetlać stringi
      sbi DC_PORT,DC_PIN    ; ustawienie trybu DATA
   ldi r25,6                ; 6 bajtów/znak
      mul r24,r25           ; oblicz offset w generatorze znaków
      ldi r30,lo8(ASCIITAB) ; adres generatora znaków
      ldi r31,hi8(ASCIITAB)
      add r30, r0           ; dodaj offset
      adc r31, r1
LCD_CHAR_1:  lpm r24,Z+     ; załaduj bajt z generatora
      rcall  lcd_wr         ; wyślij przez SPI (procedura z listingu 1)
      dec r25               ; zmniejsz licznik bajtów
      brne   LCD_CHAR_1     ; kolejny bajt
      pop r31
      pop r30
      ret                   ; koniec


lcd_gotoXY:                 ; R22 = pozycja X, R24 = pozycja Y
      cbi DC_PORT,DC_PIN    ; ustawienie trybu COMMAND
      ori r24,0x40          ; dodaj kod komendy "Set Y address"
      rcall  lcd_wr         ; wyślij do LCD
      ldi r24,6
      mul r24,r22           ; pomnóż pozycję X przez 6
      mov r24,r0
      ori r24,0x80          ; dodaj kod komendy "Set X address"
      rcall  lcd_wr
      sbi DC_PORT,DC_PIN    ; przywrócenie trybu DATA
      ret                   ; koniec

ASCIITAB:
#include „ascii.def"        ; dołączenie generatora znaków z zewnętrznego pliku.
```

Wyświetlanie bitmap

Kolejną możliwością, jaką daje nam graficzny wyświetlacz, jest wyświetlanie obrazków w postaci bitmap lub animacji składających się z sekwencji następujących po sobie obrazków. Wyświetlanie pełnoekranowych lub zajmujących całą wysokość ekranu najlepiej jest przeprowadzić w trybie adresowania pionowego. Jeśli chcemy, żeby obrazek zajmował fragment ekranu, lecz miał pełną szerokość, to lepiej jest zastosować adresowanie poziome. Samo wyświetlenie obrazka jest banalne, bo po ustawieniu wymaganej pozycji poprzez zapis liczników kolumn i wierszy wystarczy wysłać przygotowane wcześniej dane bitmapy poprzez SPI prosto do pamięci obrazu kontrolera. Ze względu na objętość kodu preferowane jest tutaj wcześniejsze przekonwertowanie standardowej bitmapy, tak aby uzyskać plik, który wysyłamy do LCD bez jakichkolwiek modyfikacji. Sposób konwersji zależy od wybranego trybu adresowania.

Przy wyborze adresowania pionowego można przekonwertować obrazek, robiąc jego odbicie lustrzane i obrót o 90 stopni w lewo. Tak zapisanej bitmapy niestety nadal nie można wysłać bezpośrednio, ponieważ posiada ona nagłówek charakterystyczny dla plików BMP oraz 86*8 bajtow danych zamiast spodziewanych 84*6. Wiec z każdych 8 bajtów opisujących jedną kolumnę, począwszy od offsetu 0x3e (początek właściwych danych), wysyłamy tylko 6 bajtów, pomijając następne 2.

Przy wyborze adresowania poziomego przekształcenie bitmapy w prawidłowy ciąg danych jest nieco bardziej skomplikowane. Nie będę dokładnie opisywał algorytmu, lecz posłużę się gotowym programem, który napisałem swego czasu w Turbo Pascalu na własne potrzeby, a który wraz z źródłami znajduje się na

List. 3. Procedura wysyłająca plik graficzny na LCD

```
lcd_image:                  ; wyświetlenie obrazka na lcd
      cbi DC_PORT,DC_PIN
   ldi r24,0x40             ; zeruj licznik kolumn (pozycja x=0)
   rcall  lcd_wr
      ldi r24,0x80          ; zeruj licznik wierszy (pozycja y=0)
   rcall  lcd_wr

      sbi DC_PORT,DC_PIN
      ldi r21, 252          ; tu wstawiamy ilość bajtów / 2 do wysłania
                            ; w zależności od wysokości obrazka
                            ; 8->42, 16->84, 24->126, 32->168, 40->210
                            ; 48->252
      ldi r30,lo8(IMAGE)    ; adres danych obrazka w pamięci FLASH
      ldi r31,hi8(IMAGE)
SENDIMG:
      LPM r24,Z             ; pobierz bajt z flash
   rcall  lcd_wr            ; wyślij bajt do LCD
   adiw   r30,1             ; zwiększ adres
      LPM r24,Z             ; pobierz bajt z flash
   rcall  lcd_wr            ; wyślij bajt do LCD
   adiw   r30,1             ; zwiększ adres
      dec r21               ; zmniejsz licznik
      brne   SENDIMG
      ret

IMAGE:
#include „IMAGE.DAT”        ; zawartość obrazka
```

CD-EP7/2003B. Aby przekonwertować bitmapę o rozmiarach 84 piksele w poziomie i 8, 16, 24, 32, 40 lub 48 pikseli w pionie wystarczy uruchomić program *bmpconv.exe*, podając jako parametr nazwę pliku do konwersji. Pamiętajmy, że jest to program DOS-owy, więc długość nazwy pliku nie może przekraczać 8 znaków + 3 znaki rozszerzenia. W wyniku działania programu otrzymamy plik binarny *out.bin*, który można załadować prosto do pamięci wyświetlacza. Przykładową procedurę wysyłającą tak przygotowany plik pokazano na **list. 3**. Na początku zerowane są liczniki kolumn i wierszy poprzez wysłanie do kontrolera komend 0x40 i 0x80 (przy niskim poziomie na D/C), co oznacza, że będziemy wyświetlać obraz począwszy od lewego górnego rogu ekranu. Następnie do licznika pętli wpisujemy potrzebną liczbę bajtów do wysłania podzieloną przez 2, a to ze względu na to, aby licznik mógł się zmieścić w pojedynczym 8-bitowym rejestrze. Dalej do rejestru Z (R31:R30) wpisujemy adres komórki pamięci Flash, od którego rozpoczynają się dane obrazka. No i pozostała już tylko pętla pobierająca kolejne bajty z pamięci Flash i wysyłająca je do LCD poprzez interfejs SPI. W każdym obiegu pętli wysyłane są 2 bajty oraz zwiększany jest wskaźnik danych - rejestr Z. Jeśli chcielibyśmy wysyłać obrazek z pamięci RAM, zamiast z pamięci Flash, wystarczy instrukcję

```
LPM r24, Z
```

zamienić na

```
LD r24, Z+;
```

oraz pominąć instrukcję

```
adiw r30,1
```

ponieważ można tu wykorzystać automatyczne zwiększanie wskaźnika w instrukcji `LD r24,Z+`.

Bargrafy, czyli paski postępu

Kolejnym elementem, który możemy wyświetlić na naszym LCD, jest pasek postępu pokazujący procent wykonania jakiejś operacji. Pasek ten będzie umieszczony na wyświetlaczu poziomo, podobnie jak jedna linijka tekstu, więc będziemy pracować w trybie adresowania poziomego. Ta procedura jest równie prosta jak poprzednie. Najpierw musimy określić miejsce, gdzie będzie się zaczynał nasz pasek postępu, czyli standardowo zapisać dane do liczników kolumn i wierszy. Następnie wywołujemy pokazaną na **list. 4** procedurę *lcd_bar*, podając w rejestrze R24 stopień zapełnienia (długość zapełnionego paska w pikselach), a w rejestrze R22 ogólną długość całego bargrafu. Uaktualnienie pozycji paska polega na kolejnym wywołaniu tej procedury ze zmienioną zawartością rejestru R24 wraz z wcześniejszym ustawieniem pozycji paska na taką jak za pierwszym razem.

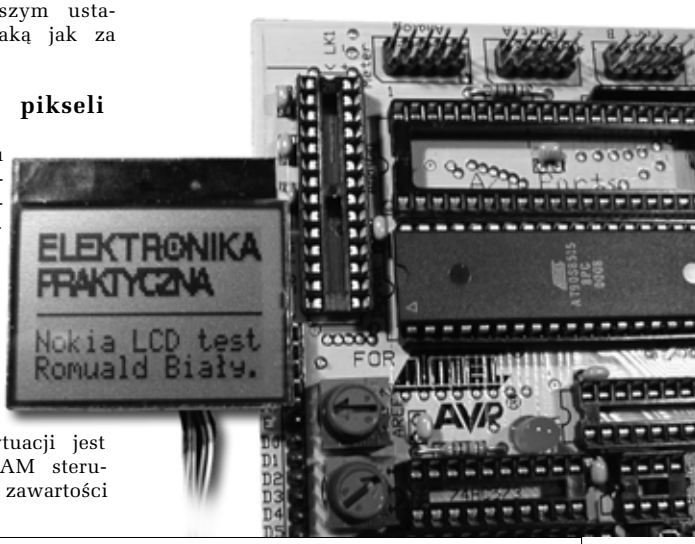
Zapalanie pojedynczych pikseli i rysowanie linii

Niestety w tym miejscu kończy się wszystko co krótkie, łatwe i przyjemne, a zaczynają się prawdziwe „schody”. I to z dwóch powodów. Po pierwsze, aby zapalić pojedynczy piksel, musimy zmienić jeden bit z bajtu odwzorowującego 8 pionowych pikseli, a nie możemy odczytać z pamięci wideo wyświetlacza poprzedniej jego zawartości. Jedynym wyjściem z sytuacji jest przechowywanie w pamięci RAM sterującego mikrokontrolera kopii zawartości pamięci wideo wyświetlacza, operowanie na jego zawartości i późniejsza aktualizacja zawartości pamięci wideo wyświetlacza. Kopia pamięci wideo zajmuje dość dużo miejsca - 504 bajty pamięci RAM mikrokontrolera - czyli prawie całą dostępną wewnętrzną pamięć mikrokontrolera AT90S8515 lub połowę pamięci z ATmega8 czy ATmega161. Oczywiście podłączając do naszego mikrokontrlera zewnętrzną pamięć RAM lub stosując jeden z wyższych modeli ATmega posiadający 2 lub 4 kB wewnętrznego RAM-u, pozbywamy się tego kłopotu.

Pozostaje drugi problem, czyli ilość i czasochłonność wymaganych do tego celu obliczeń. Do datkowo synchronizacja zawartości kopii pamięci wideo z zawartością pamięci kontrolera wyświetlacza jest również czasochłonna przy założeniu każdorazowej aktualizacji zawartości całego wyświetlacza lub skomplikowana przy próbie wybiórczej aktualizacji fragmentów pamięci wideo. Najlepszym wyjściem z omawianej sytuacji będzie napisanie odpowiednich procedur w języku wyższego poziomu, czyli np. w C.

Uniwersalny sterownik LCD z Nokii

Jeden z członków forum dyskusyjnego poświęconemu odtwarzaczom MP3 *yampp* napisał swoją wersję uniwersalnego sterownika wyświetlaczy LCD z Nokii, posiadającą możliwość zapalania i gaszenia pojedynczych pikseli, rysowania linii oraz wyświetlania znaków i całych *stringów*. Program ten przeznaczony jest do skompilowania bezpłat-



List. 4. Procedura odpowiadająca za wyświetlenie bargrafu

```
lcd_bar:                        ; wyświetlenie paska postępu
   mov r25,r24
      ldi r24,0b01111111        ; wygląd początku paska
      sbi DC_PORT,DC_PIN        ; ustaw tryb wysyłania danych
      rcall  lcd_wr             ; narysuj początek bargrafu

LCD_BAR_2:    ldi r24,0b01011101    ; wygląd zapełnionego fragmentu paska
      tst r25                   ; sprawdź czy rysujemy zapełniony
      brne   LCD_BAR_1          ; skok jeśli tak
      ldi r24,0b01000001        ; załaduj wygląd pustego paska
      inc r25                   ; korekta r25
LCD_BAR_1:    dec r25           ; zmniejsz licznik zapełnionych
      rcall  lcd_wr             ; wyślij R24 do LCD
      dec r22                   ; zmniejsz licznik długości paska
   brne   LCD_BAR_2             ; pętla

      ldi r24,0b01111111        ; wygląd końca paska
      rjmp   lcd_wr             ; wyślij do LCD i wyjdź z procedury
```

nym kompilatorem AVR-GCC i przeznaczony dla mikrokontrolera ATmega8. Oczywiście może zostać zaadaptowany do umieszczenia w innym typie mikrokontrolera AVR, byle by posiadał odpowiednią ilość pamięci RAM. Kod źródłowy tego sterownika publikujemy na CD-EP7/2003B.

Postaram się pokrótce przedstawić jego możliwości i sposób wywoływania zawartych w nim funkcji. Sterownik składa się z pliku nagłówkowego *NokiaLCD.h*, zawierającego deklaracje kilku wymaganych stałych, definicję sposobu podłączenia wyświetlacza do mikrokontrolera - czyli deklarację wykorzystanych pinów mikrokontrolera, oraz prototypy dostępnych funkcji. Właściwy kod programu znajduje się w pliku *NokiaLCD.c* i zawiera wszystkie procedury niezbędne do obsługi wyświetlacza wraz z inicjalizacją i obsługą interfejsu SPI oraz tablicę generatora znaków.

Oto opis dostępnych funkcji sterownika:

- **LcdInit()** - służy ona do zainicjowania interfejsu SPI mikrokontrolera oraz podłączonego wyświetlacza. Po zakończeniu działania tej funkcji wyświetlacz ma odpowiednio ustawione rejestry wewnętrzne, znajduje się w trybie adresowania poziomego - ponieważ wszystkie funkcje korzystają właśnie z tego trybu. Inicjalizacja zeruje zawartość cienia pamięci wideo jak również i sam wideo-RAM wyświetlacza. Powinna być uruchomiona jako jedna z pierwszych funkcji całego programu, ponieważ ustawia funkcje pinów mikrokontrolera oraz podaje prawidłowy sygnał RESET do wyświetlacza.
- **LcdClear()** - wyzerowanie zawartości cienia pamięci wideo. Aby wyczyścić również zawartość wyświetlacza, należy po *LcdClear()* wywołać funkcję *LcdUpdate()*.
- **LcdUpdate()** - powoduje aktualizację zawartości pamięci wideo wyświetlacza zgodnie z zawartością cienia w mikrokontrolerze. Aktualizowany jest tylko obszar, w którym były dokonane jakiekolwiek zmiany od czasu poprzedniej aktualizacji.
- **LcdContrast(byte contrast)** - ustawienie kontrastu wyświetlacza. Wartość bajtu contrast musi zawierać się pomiędzy 0x00 a 0x7f, lecz jak napisałem wcześniej, użyteczny zakres wynosi od 0x20 do 0x58.
- **LcdGotoXY(byte x, byte y)** - ustawienie pozycji wyświetlania na LCD. Koordynaty bazowane są na czcionce o podstawowej wielkości, czyli zakres X wynosi od 1 do 14, a zakres Y od 1 do 6. Funkcja ta nie zmienia zawartości liczników kolumn i wierszy, a jedynie ustawia odpowiedni offset aktualnego adresu w kopii pamięci wideo.
- **LcdChr(LcdFontSize size, byte ch)** - wyświetlenie znaku ASCII o wielkości określonej parametrem size w miejscu aktualnej pozycji wyświetlania. Mamy do dyspozycji dwie wielkości czcionki. Standardową 5x7 punktów, którą określa stała *FONT_1X* lub wartość 1, oraz podwójnej wielkości, pogrubioną czcionkę oznaczoną stałą *FONT_2X* lub wartością 2. Funkcja ta automatycznie zwiększa aktualną pozycję wyświetlania. Aby zobaczyć zmiany, musimy po niej wywołać funkcję *LcdUpdate()*.
- **LcdStr(LcdFontSize size, byte *dataPtr)** - wyświetlenie stringu z pamięci RAM mikrokontrolera zaczynającego się od adresu dataPtr. Pozostałe parametry jak dla funkcji *LcdChr*.
- **LcdPixel(byte x, byte y, LcdPixelMode mode)** - wyświetlenie pojedynczego piksela. Parametry X i Y określają pozycję na wyświetlaczu, a *mode* oznacza tryb pracy. Wartość X musi się mieścić w przedziale od 0 do 83, a Y w przedziale od 0 do 47. W zależności od trybu, piksel na danej pozycji może zostać zapalony, zgaszony lub zmieniony - czyli jeśli był wcześniej zapalony, to zostanie zgaszony i na odwrót. Stałe opisujące tryb to odpowiednio *PIXEL_ON*, *PIXEL_OFF* i *PIXEL_XOR*. Aby zobaczyć zmiany, musimy po niej wywołać funkcję *LcdUpdate()*.
- **LcdLine(byte x1, byte y1, byte x2, byte y2, LcdPixelMode mode)** - narysowanie linii zaczynającej się w punkcie o współrzędnych *x1,y1* i kończącej się w punkcie o współrzędnych *x2,y2*. Dopuszczalne wartości pozycji, jak również tryb mode - tak jak dla funkcji *LcdPixel*. Aby zobaczyć zmiany musimy po niej wywołać funkcję *LcdUpdate()*.

Pozostałe funkcje widoczne wewnątrz pliku *NokiaLcd.c* są wewnętrznymi funkcjami sterownika.

List. 5. Procedura obsługi programowego interfejsu SPI napisana w asemblerze 8051

```
; P1.4 -> SCE
; P1.5 -> MISO - nieużywany dla LCD
; P1.6 -> MOSI czyli SDIN wyświetlacza
; P1.7 -> SCLK

LCD_WR:   CLR P1.4          ; SCE = L
   MOV R2,#08H              ; licznik, 8 bitów do wysłania z ACC
SPIL:     RLC A             ; przesuń bit do carry
   MOV P1.6,C               ; wystaw carry na pin P1.6 (SDAT)
   SETB   P1.7              ; SCLK = L
   NOP
   CLR P1.7                 ; SCKL = L
   MOV C,P1.5               ; Pobranie bitu z magistrali - zbędne
                            ; w przypadku współpracy z LCD
   DJNZ   R2,SPIL           ; zmniejsz licznik i wróć do pętli
   SETB   P1.4              ; SCE = H
   RET
```

Rys. 10

Nokia LCD + PC? Czemu nie!

Zapewne wielu Czytelników zainteresowanych wykorzystaniem opisanych wyświetlaczy będzie chciało szybko sprawdzić jego działanie bez wcześniejszego budowania i oprogramowywania całego systemu opartego na mikrokontrolerze. Jest na to prosta, a przy tym skuteczna metoda. Podłącz swój wyświetlacz bezpośrednio do złącza drukarkowego w swoim PC. No prawie bezpośrednio, ponieważ sygnały logiczne na złączu LPT komputera są zgodne ze standardem TTL 5V, a wyświetlacz potrzebuje napięć o wartości 3,3 V. Konieczny będzie więc konwerter poziomów zbudowany na przykład w oparciu o układ 74LVC245 (jego schemat publikujemy na CD-EP7/2003B, w programie *Parallel to SPI*). Do tego odpowiednie oprogramowanie, zasilacz 3,3 V i już możemy podziwiać działający wyświetlacz.

Na CD-EP7/2003B publikujemy program napisany przez kolejnego użytkownika forum dotyczącego *yamppa* (to naprawdę jest kopalnia wiedzy), umożliwiający sterowanie takim wyświetlaczem z poziomu aplikacji MS Windows (**rys. 10**). Program ten nie posiada zbyt rozbudowanych funkcji, lecz pozwala na zainicjowanie wyświetlacza, regulowanie kontrastu, ustawienie odpowiedniego trybu, wyświetlanie tekstów oraz konwersję i wyświetlanie obrazków. Dodatkowo pozwala on na tworzenie, edycję i zapisywanie wzorów czcionek generatora znaków w postaci definicji *db* do późniejszego wykorzystania we własnych programach. Kolejną możliwością tego programu jest sterowanie podłączonego w podobny sposób dekodera MP3, jakim jest układ VS1001, a więc odtwarzanie plików MP3. Schemat podłączenia wyświetlacza LCD lub VS1001 uzyskamy po wybraniu z menu *View* opcji *Schematics...*.

Na zakończenie, życząc wszystkim Czytelnikom udanych eksperymentów z opisywanymi wyświetlaczami, przedstawiam zapowiedzianą wcześniej procedurę obsługi programowego interfejsu SPI napisaną w asemblerze mikrokontrolera z rodziny 8051 - czyli np. na tzw. „małego“ AT89C2051 (**list. 5**). Bajt do wysłania ładujemy do akumulatora i wywołujemy funkcję przez *ACALL LCD_WR*. Jeśli zastosujemy ją do obsługi innego układu komunikującego się poprzez SPI, to po powrocie w akumulatorze otrzymamy bajt odczytany z tego układu. Ponieważ kontroler naszego LCD nie ma takiej możliwości, zawartość akumulatora możemy zignorować lub nawet usunąć instrukcję *MOV C,P1.5*.

Romuald Biały