

W kilku poprzednich numerach EdW miałeś okazję dowiedzieć się wielu ciekawych informacji na temat układów mikroprocesorowych, czyli w skrócie "mikroprocesorów". Wiesz już że te, dość złożone w budowie, elementy elektroniczne stosowane są powszechnie w komputerach klasy PC. W nie tak odległej przeszłości wielu użytkowników sędziwego dziś Spektrusia, Commodora czy Atari często nie zdawało sobie sprawy, że jest posiadaczem mniej czy bardziej skomplikowanego układu mikroprocesorowego. Jednak mikroprocesory nie zostały wynalezione jedynie po to, aby zadowalać coraz bardziej wymagającego użytkownika, żadnego maszyn - coraz szybciej obrabiających dane, chcącego mieć dostęp do multimedialnych gier zajmujących niebotyczne ilości miejsca na dysku twardym komputera domowego.

Od samego początku inteligentnych układów cyfrowych na rynku elektronicznym istniała grupa dość prostych, na pierwszy rzut oka, mikroprocesorów, których ewolucja nie potraktowała tak ostro, jak to miało miejsce w wypadku rodziny 8086. Jeżeli nie wiesz, co kryje się pod tą nazwą, przypomnę ci, że układy 8086 to prawdziwi "pradziadowie" procesorów Pentium obecnie masowo stosowanych w komputerach PC.

Wspomniane układy, będące niejako oddzielną gałęzią w rodzinie układów cyfrowych wielkiej skali integracji (podobnie jak "małpy" w teorii Darwina), przetrwały w niezmienniej postaci od kilkunastu lat. Co mogło być powodem tego stanu rzeczy? Otóż dzięki architekturze, czyli budowie wewnętrznej tych układów, okazało się możliwe zastosowanie ich nie tylko w specjalizowanym sprzęcie komputerowym. Głównym rynkiem zbytu okazali się producenci różnego rodzaju sprzętu gospodarstwa domowego, od ekspresów do kawy począwszy, poprzez sprzęt radio-telewizyjny, AGD, na motoryzacji skończywszy.

Przy okazji lektury artykułów w EdW na temat mikroprocesora dowiedziałeś się, że sam mikroprocesor to nie wszystko. Nasz na pozór inteligentny układ cyfrowy bez dołączenia kilku dodatkowych elementów zewnętrznych: zegara, pamięci, układów wejścia/wyjścia (I/O) potrafi niewiele.

I wtedy ktoś wpadł na pomysł umieszczenia samego mikroprocesora z wymienionymi układami peryferyjnymi z jednym układzie scalonym. Tak powstał pierwszy "mikrokontroler" a właściwie "mikrokomputer jednoukładowy".

Słowo "mikrokomputer" nie jest bynajmniej na wyrost, bowiem stworzony scalak był w istocie kompletnym komputerem tylko że w małym formacie. Wewnątrz posiadał jakby rdzeń, który

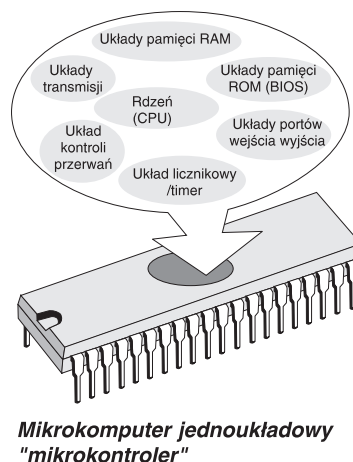
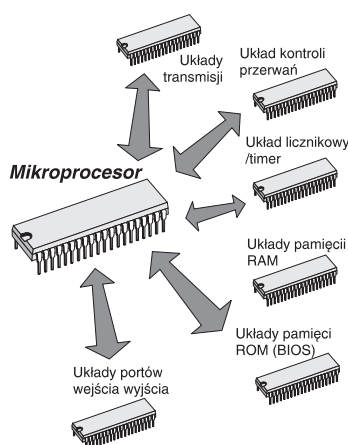


potrafił przetwarzać komendy wydawane przez programistę; pamięć - w której mógł przechowywać wyniki obliczeń oraz układy do komunikacji ze światem zewnętrznym - czyli tzw. porty. Tak funkcjonalna budowa oraz, co miało nie małe znaczenie, niska cena mikrokontrolera, uutorowała mu drogę do zastosowań praktycznie wszędzie.

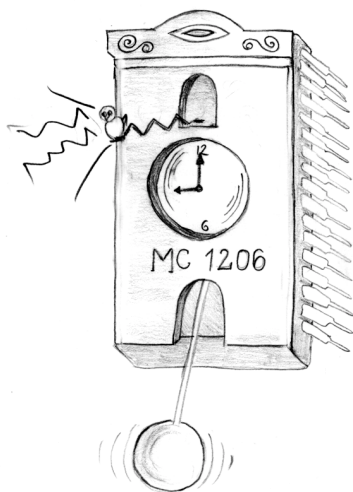
Rysunek 1 obrazuje różnicę między "mikroprocesorem" a "mikrokomputerem" (mikrokontrolerem, jak kto woli).

Widać, że wszystkie urządzenia peryferyjne znajdujące się "na zewnątrz" mikroprocesora, w przypadku mikrokomputera zostały umieszczone w jednym układzie scalonym. I w tym tkwi potęga naszych prostych mikrokomputerów.

W naszych dalszych rozważaniach, pomimo że mowa będzie o mikrokomputerach jednoukładowych, będziemy zamiennie używać określeń "mikrokontroler" lub nawet "mikroprocesor". Zapa-



Rys. 1. Mikroprocesor do pracy potrzebuje wielu dodatkowych układów peryferyjnych, a typowy "mikrokontroler jednoukładowy" ma je wbudowane w strukturę.



miętaj to, żebyś się nie pomylił. Ze względów stylistycznych będziemy używać nawet określenia "mikroprocesor", choć nie jest to do końca ściśle. Ale przecież już wiesz, drogi Czytelniku, o czym będzie mowa.

I tak niektóre z mikrokontrolerów wyspecjalizowały się w konkretnych dziedzinach tak bardzo, że nie potrafiły znaleźć miejsca gdzie indziej. Najprostszym przykładem niech będzie wspomniany już układ zegara MC1206. Któż z was nie próbował, a przynajmniej nie słyszał o tym jakże popularnym, szczególnie na giełdach elektronicznych, układzie cyfrowym. Ten "zegarek" był przecież mikrokontrolerem, tylko potrafiącym wykonywać określone czynności związane z pomiarem czasu. Układ oscylatora miał, prawda? Pamięć wewnętrzną (np. alarmu) też, wyjścia do sterowania wyświetlaczami LED (porty I/O) także, więc teraz mi chyba nie zarzucisz, drogi Czytelniku, że ta kostka to nie był prosty ale funkcjonalny mikrokontroler.

Pomyśl teraz, czy mając te wszystkie elementy składowe, zamiast np. wyświetlaczy LED nasza kostka MC12... mogłaby pracować w roli programatora do pralki automatycznej. Niestety, w czasach PRLu nikt o o tym nie pomyślał, a w każdym razie nie doczekano się wdrożenia takiego układu. Powstał natomiast prymitywny, elektromechaniczny programator, którego kolejny, regenerowany egzemplarz pracuje w 20-letniej pralce autora (używanej raczej ze względów sentymentalnych).

Przykładów może być wiele, my jednak zajmiemy się jednokładowcami bardziej uniwersalnymi z twego punktu widzenia - takimi, które będziesz sam mógł "zmusić" do wykonywania okreś-

lonych czynności w zbudowanym przez ciebie układzie.

Będziesz mógł zrobić sobie swój własny MC1206, lecz np. z 25 alarmami, ze sterowaniem 4 przełącznikami, stoperem. W przypływie nudy wykorzystasz ten sam układ scalony - mikrokontroler i zbudujesz z pomocą kilku dodatkowych elementów dyskretnych miernik częstotliwości lub licznik obrotów silnika do twego samochodu. Wreszcie dając upust narastającej górze pomysłów wykorzystasz mikrokontroler do budowy przemysłowego systemu alarmowego ze zdalnym sterowaniem wszystkich funkcji w twoim mieszkaniu: od gaszenia światła począwszy, na sygnalizacji przecieku wody lub gazu skończywszy.

Za trudne? Nic podobnego, znam to z autopsji. W czasach szkoły średniej (lata 80), nie miałem zielonego pojęcia o mikroprocesorach, nie mówiąc o tym, że kupienie odpowiedniej kostki było nie tylko ładą trudnością ale i... głupotą ze względu na kompletny brak jakiegokolwiek literatury na temat projektowania układów z wykorzystaniem mikroprocesorów. Moją pasją była technika cyfrowa czyli najczęściej sklejanie z wielu kostek TTL czy CMOS jakiegoś sensownie działającego układu, który często po dłuższym lub krótszym okresie czasu odchodził do lamusa, czyli krótko mówiąc kończył w kartonie z innymi elektronicznymi śmieciami, czekając, że może któraś z kostek przyda się w przyszłości.

I wtedy pojawił się ON - mikrokomputer jednokładowy. Oczarował mnie bardziej niż pocziwy PC XT, ze względu na swoją prostotę i możliwość wielu zastosowań.

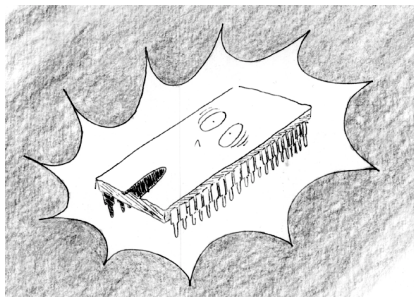
Tak zamieniłem płytkę drukowaną zegara o wymiarach 20x30 cm z kilkunastoma układami scalonymi (całość pobierała ponad 0,5A prądu) na zgrabny układ mieszczący się w niewielkiej i estetycznej obudowie wielkości dużego pudełka po zapalkach. Rodzina i ja byli dumni i zadowoleni z urządzenia, które funkcjonalnością a zarazem wielkością wyświetlaczy konkurowało z tymi oferowanymi w ówczesnych czasach na rynku. A potem sprawy potoczyły się szybko, po kolei na warsztat poszedł mój pierwszy amplituner z RADMORu, potem wymieniłem wnętrze w zegarze akwariowym, także nieodwracalnej modyfikacji uległa moja, z trudem kupiona, "szuflada" z DIO-RY. I wszystko spisuje się do dziś dzień znakomicie!

Jeżeli w tym momencie, drogi Czytelniku, pomyślisz: No tak, tylko mikroprocesor, a co zrobić z szufladą niepotrzebnych TTLi i CMOSów?. Odpowiem ci: zatrzymaj je wszystkie. Układy mikroprocesorowe nie kończą się na... mikroprocesorach! Twoje zapasy z pewnością zostaną z pożytkiem wykorzystane, bardziej racjonalnie i ekonomicznie zarazem, w wielu układach elektronicznych jako peryferia samego mikrokontrolera.

Po tych kilku westchnięciach za minioną epoką wróćmy jednak do konkretnych na temat mikrokomputerów jednokładowych. Jak powiedziałem wcześniej, układy te istnieją do dziś, a ich różnorodność i możliwości zastosowań są nieograniczone.

Obecnie istnieje kilka rodzin tych układów, których producentami są największe koncerny elektroniczne na świecie.





W mikroprocesorowym świecie najbardziej znani producenci to:

- Microchip, ze swoją rodziną "jednouladkowców" PIC...
- Motorola, lansująca układy 8,16 i 32-bitowych mikrokontrolerów jednouladkowych
- Intel, produkujący bodaj najbardziej popularne procesory serii 8051...
- Zilog, producent nowoczesnych kontrolerów jednouladkowych - następców pocziwego Z80 (wykorzystywanego w produkcji sędziwych ZX81, ZX Spectrum)
- SGS-Thompson z rodziną ST62.

Istnieje także kilka innych firm, które na bazie licencji opracowały mutacje tych procesorów, wyposażając je w wiele dodatkowych bloków funkcjonalnych, zachowując przy tym pełną kompatybilność ze swymi pierwowzorami. Do nich z pewnością należy zaliczyć Philipsa, Siemensa oraz dwie amerykańskie firmy: Dallas oraz Atmel, które w ostatnich latach zaskoczyły projektantów kilkoma udanymi wersjami najbardziej popularnych mikrokontrolerów jednouladkowych.

Do zastosowań amatorskich (a nawet w pełni profesjonalnych) najbardziej praktyczne są mikroprocesory 8-bitowe. Wersje 16 i 32 bitowe są po prostu za dobre, a także za drogie jak na potrzeby domowego czy szkolnego laboratorium.

Wśród popularnych "8-bitowców" do niedawna prym wiodły Z80, niestety ze względu na ograniczone możliwości obsługi urządzeń peryferyjnych (wejścia/wyjścia) bez konieczności stosowania dodatkowych układów scalonych rodziny Z80 słuch praktycznie o nich zaginęły. Obecnie najbardziej popularne mikrokontrolery to kostki PIC (Microchip) oraz rodzina MCS-51, czyli procesory oparte o układ 8051.

I właśnie te ostatnie, drogi Czytelniku, zostaną opisane w kolejnych numerach EdW.

Dlaczego akurat te? Odpowiedź jest prosta. Po pierwsze: są to najłatwiej dostępne i najtańsze (w stosunku ceny do możliwości) układy mikroprocesorowe na rynku. Po drugie, wszędzie roi się od shareware'owych programów na ich temat, a na naszym rynku zaczęły się poja-

wiać podręczniki podejmujące temat mikrokontrolerów 8051 i pochodnych. Wreszcie dostępność w miarę tanich narzędzi do wspomagania projektowania przeważała na ich korzyść. Tylko nie myśl od razu, że w cyklu poświęconym 8051 będziemy cię zmuszać do kupowania komputera, programów czy nawet książek. Nie! Do zaznajomienia się z możliwościami tych procesorów nie będzie ci nawet potrzebny komputer! Tak, to jest możliwe.

Przyjrzyjmy się teraz, co zawiera typowy przedstawiciel rodziny MCS-51 - mikrokomputer jednouladkowy 8051 (rys. 2). Jak widać, w jednej kostce zawarto wszystkie niezbędne do pracy układy, toteż wystarczy dosłownie kilka biernych elementów zewnętrznych aby ruszyć do pracy. Ale jakiej? O tym dowiesz się w kolejnej części naszego cyklu o 51-ce.

W jednym układzie scalonym zawarto:

- rdzeń mikroprocesora CPU z 8-bitową jednostką arytmetyczno-logiczną (ALU), zdolna do wykonywania obliczeń na liczbach 8-bitowych;
- uniwersalne dwukierunkowe porty wejścia/wyjścia, do komunikowania się ze światem zewnętrznym po poprzez zapisywanie do nich jak i odczyt przez nie danych cyfrowych (w niektórych odmianach 8051 z wbudowanymi przetwornikami A/C i C/A, także wielkości analogowych;
- programowany szeregowy port transmisji dwukierunkowej, który może np. służyć do komunikowania się z dowolnym komputerem wyposażonym w złącze RS232C;
- dwa (w innych wersjach 3) uniwersalne liczniki/timery, do dowolnego wykorzystania;
- układ generowania przerwań systemowych, zawierający także możliwość generowania przerwań zewnętrznych;
- układ wewnętrzny oscylatora, który ogranicza do minimum konieczność stosowania zewnętrznych elementów do pojedynczego rezonatora kwarcowego oraz dwóch dodatkowych kondensatorów ceramicznych;
- wreszcie pamięć do przechowywania danych i wyników obliczeń: RAM;
- oraz wewnętrzna pamięć typu ROM, w której zawarty jest program działania mikrokontrolera.

Program działania jest tworzony przez konstruktora w procesie tworzenia aplikacji, a następnie jest zapisywany za pomocą programatora w strukturę mikrokontrolera. Ponadto mikroprocesor 8051 posiada możliwość dołączenia z zewnątrz dodatkowych układów pamięci statycznych RAM (do przechowywania danych) oraz pamięci EPROM/ROM z której może odczytywać polecenia -

czyli program. W tym ostatnim przypadku często wewnętrzna pamięć ROM jest wtedy nieaktywna, lub nie ma jej wcale, ale o tym później.

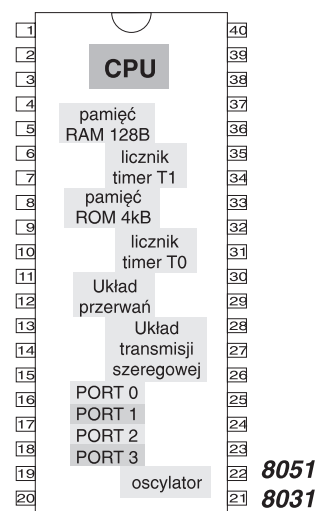
Na rynku istnieje kilka podstawowych wersji procesora 8051. Wszystkie one mają takie same wewnętrzne bloki funkcjonalne, różnica polega na rodzaju pamięci programu - czyli pamięci stałej z której mikrokontroler pobiera rozkazy.

Jak wiesz, wśród pamięci stałych ROM najbardziej popularne są pamięci EPROM (programowane elektrycznie, kasowane promieniami UV) głównie ze względu na ich cenę. Rzadziej stosowane są pamięci ROM programowane przez producenta lub pamięci EPROM/OTP (OTP - One Time Programmable) - czyli jednokrotnie programowane pamięci EPROM (jednokrotnie - bo nie ma możliwości ich kasowania promieniami UV, co jest wynikiem braku okienka kwarcowego w obudowie pamięci).

Ostatnio coraz większą popularność zdobywają pamięci EEPROM, czyli pamięci które można programować jak i kasować elektrycznie. Przy stosowaniu pamięci EEPROM odpada konieczność stosowania kłopotliwych i często drogich kasowników pamięci EPROM (z lampą ultrafioletową, a pamięć można programować wielokrotnie, nawet 100 tysięcy razy).

W różnych wersjach procesorów 8051 stosuje się różne, wymienione wcześniej typy pamięci programu. Tak więc mamy mikrokontrolery w wersji ROM, OTP, EPROM, wreszcie EEPROM oraz wersje pozbawione pamięci programu w ogóle, przystosowane do pracy z dołączoną z zewnątrz dowolną pamięcią programu.

I tak, w zależności od wbudowanej w układ mikrokontrolera 8051 pamięci



Rys. 2. Mikrokomputer 8051.

Też to potrafisz

programu, producenci ustalili w miarę jednolite i przejrzyste symbole, których znajomość (na razie teoretyczna) z pewnością przyda się w późniejszych zakupach tych kostek.

W **tabeli 1** zestawiono oznaczenia mutacji procesora 8051, oraz krótką charakterystykę zastosowanej pamięci programu. Literka "C" w nazwie każdego z nich oznacza, że każdy układ wykonany jest w wersji CMOS. Niegdyś dość popularne były wersje HMOS (bez literki C, np. 8031), lecz jest to przeszłość, toteż nie będziemy się nimi zajmować.

Jeżeli przez przypadek natrafisz, np. w BOMISie, na układ w wersji HMOS i możesz go nabyć za grosze, skorzystaj i kup go. Taki układ jest identyczny jak w wersji CMOS, lecz będzie pobierał więcej prądu podczas pracy, co często nie jest problemem.

Ze względu na rzadkość takich sytuacji, w swoich rozważaniach będziemy podawać dane i parametry techniczne dotyczące układów 8051, wykonanych w wersji CMOS.

Podane w tabeli wersje procesorów 80C51 i 80C52 to typowe "odpady" produkcyjne wielkich koncernów produkujących sprzęt elektroniczny. "Odpady" to nie znaczy bezwartościowe lub wybrakowane. W pamięci wewnętrznej układu 8051 producent zapisał jakiś program dla konkretnego odbiorcy (np. wytwórcy pralek automatycznych). Kostki te nie zostały jednak sprzedane temu odbiorcy. Dla ciebie wpisany program jest bezużyteczny. Ale, jak wspomniałem wcześniej, mikrokontrolery rodziny 8051 mają możliwość pracy z wbudowaną lub zewnętrzną pamięcią programu. Jeżeli decydujemy się na wykorzystanie tej drugiej możliwości, pamięć wewnętrzną

Tabela 1

| Symbol handlowy | Opis |
|-----------------|---|
| 80C51 | wersja z wewnętrzną pamięcią programu typu ROM, której zawartość jest nieznana z naszego punktu widzenia, toteż układ możemy wykorzystać do pracy tylko z dołączoną zewnętrzną pamięcią np. EPROM do której zapiszemy nasz program (wtedy pamięć ROM jest wyłączona - nieaktywna) |
| 80C31 | wersja procesora bez wewnętrznej pamięci programu. Mikrokontroler w tej wersji może pracować tylko z dołączoną zewnętrzną pamięcią jak dla 80C51. |
| 87C51 | wersja z wbudowaną pamięcią EPROM. Obudowa mikroprocesora posiada okienko kwarcowe, dzięki któremu możliwe jest kasowanie zawartości tej pamięci, co umożliwia wielokrotne programowanie całego układu. |
| 89C51 | najnowsza wersja procesora z kasowaną elektrycznie pamięcią EEPROM. Ponieważ w tej wersji cała pamięć programu EEPROM może być kasowana bardzo szybko - za pomocą tylko 1 impulsu, procesory w tej wersji nazywa się typu "Flash" (czyt. "flesz") |
| 80C52 | jest to procesor identyczny z 8051 tyle że posiada dodatkowy trzeci programowalny licznik/timer (nazywany jako "T2") i dwa razy więcej pamięci RAM (256B). Reszta jak dla 80C51 - patrz wyżej. |
| 80C32 | jak dla 80C31 z uwzględnieniem "T2" i RAM |
| 87C52 | jak dla 87C51 z uwzględnieniem "T2" i RAM |
| 89C52 | jak dla 89C51 z uwzględnieniem "T2" - RAM |

można fizycznie odłączyć - poprzez zwarcie do masy odpowiedniego wyprowadzenia mikrokontrolera 8051. W takiej aplikacji można zatem użyć wersji 80C31 - bez wewnętrznej pamięci programu, lub bardzo taniej (nazwanej wcześniej "odpadową") wersji 80C51. W obu przypadkach działanie układu będzie takie same. Ze względu na ogromną różnicę w cenie tych dwóch wersji, powinniśmy używać tańszej kostki 80C51. Zapytasz pewnie: To po co w ogóle na rynku są wersje 80C31, skoro można użyć 80C51

po niższej cenie? Otóż pamięć pobiera prąd - różnica polega na poborze prądu przez te układy. Jednak różnica ta wynosi zaledwie kilka mA, toteż w naszych zastosowaniach nie ma to żadnego znaczenia.

W następnym odcinku przyjrzymy się temu, co "wystaje" z mikroprocesora - czyli wyprowadzeniom i ich znaczeniu dla układu samego mikrokontrolera.

Sławomir Surowiński

W poprzednim numerze "Elektroniki dla Wszystkich" zaznajomiliśmy Czytelników z pojęciem mikroprocesora i mikrokontrolera. Przedstawiliśmy ogólne założenia dotyczące budowy układów scalonych tego typu oraz przybliżony sposób współpracy z innymi układami peryferyjnymi.

W tym odcinku przedstawimy ogólny opis wyprowadzeń mikroprocesora. Jest to drugi z odcinków wstępnych z cyklu obejmującego naukę programowania procesora 8051.

Przypominamy, że wkrótce zamknijemy listę kandydatów na uczniów w "klasie mikroprocesorowej". Na zgłoszenia chętnych czekamy do końca maja. "Klasa mikroprocesorowa" to grupa 20-30 osób, które otrzymają bezpłatnie od firmy AVT zestaw edukacyjny (składający się z dwóch płytek z procesorem, klawiaturą, wyświetlaczami itp.). Osoby te zobowiązane będą do przeprowadzania wszystkich prostych ćwiczeń z zakresu nauki programowania oraz do zgłaszania autorowi cyklu wszelkich wynikłych niejasności czy problemów. Ma to na celu praktyczne sprawdzenie stopnia opanowania przedstawionego materiału, a także niewątpliwie zapewni skuteczność nauki. Do skorzystania z tej możliwości zapraszamy osoby w różnym wieku, od 12 do 80 lat.



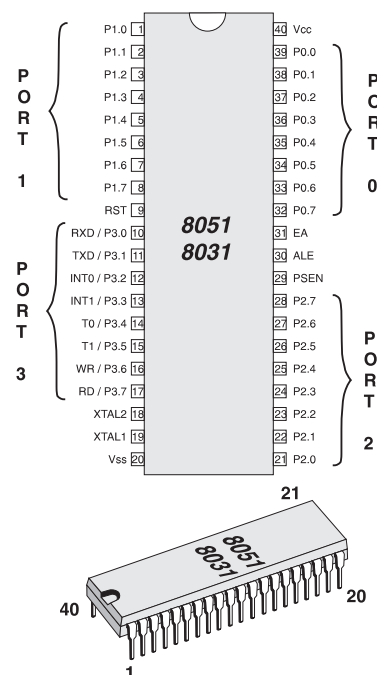
Dlaczego 8051?

Ponieważ elektronika oparta na układach mikroprocesorowych wkracza pod strzechy coraz silniej, warto by przyrzeć się bliżej jednemu, bodaj najpopularniejszemu układowi tego typu, a mianowicie mikrokontrolerowi 8051.

Zapewne wielu Czytelników EdW spotyka się z oznaczeniem 8051. Niektórych z pewnością ogarnia zimny dreszcz, inni jak wynika z listów, są zaciekawieni tematem i możliwościami programowania mikroprocesorów. Postaram się w sposób przystępny, tak merytorycznie jak i finansowo, (niestety z nauką wiąże się część praktyczna, która wymaga minimum sprzętu do nauki programowania) pokazać i przekonać Was o tym że projektowanie układów przy wykorzystaniu mikrokontrolera 8051 nie jest trudne. Wymagane są jedynie podstawowe wiadomości z techniki cyfrowej, mówiąc konkretnie każdy, kto zna podstawowe bramki logiczne oraz najprostsze typy przerzutników, a jeżeli dodatkowo wykonał sam jakiś układ lub opisany w literaturze, z pewnością nie będzie miał problemów z opanowaniem sztuki korzystania z mikroprocesora 8051.

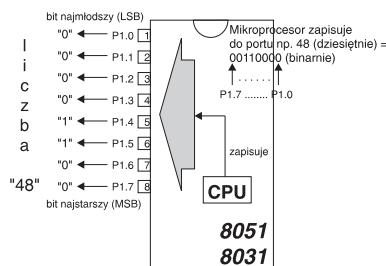
Przy opisie samego procesora, a w późniejszych numerach EdW, także podczas krótkich lekcji na temat programowania, będę za każdym razem odwoływał się do analogicznych układów wy-

konanych w standardowej technice cyfrowej (TTL czy CMOS). Dzięki temu każdy zainteresowany tematem czytelnik, zorientowany choć w podstawach cyfrowki, będzie w stanie strawić pewną porcję wiedzy, osławając się jednocześnie



Rys. 3. Opis wyprowadzeń mikrokontrolera 8051.

Też to potrafisz



Rys. 4. Zapis przykładowej liczby do portu P1.

nie z na pozór skomplikowanym układem cyfrowym, jakim jest 8051 ka.

Dla sceptyków, którzy sądzą, że programowanie procesora 8051, nawet na etapie "przedszkola", wymaga posiadania drogiego komputera klasy PC, mam miłą wiadomość. Otóż skonstruowałem podstawowy układ aplikacyjny na procesor 8051 (nieduża płytka drukowana + kilka podzespołów), dzięki któremu każda teoretyczna lekcja na łamach naszego pisma, będzie mogła być natychmiast powtórzona w praktyce na stole każdego z Was, drodzy Czytelnicy. Bynajmniej nie będzie potrzebny także żaden programator pamięci EPROM lub inne, często kosztowne wyposażenie. Wystarczą dobre chęci i trochę wolnego czasu, a z pewnością każdy z Was będzie zachwycony efektami swojej pracy, czyniąc pierwsze kroki w technice mikroprocesorowej.

Trochę o samym bohaterze

Na początek przyjrzyjmy się samemu mikroprocesorowi 8051. Warto w tym miejscu wyjąć z szuflady biurka taką kostkę, a jeżeli ktoś jej nie posiada, może ją nabyć prawie w każdym sklepie z podzespołami elektronicznymi, lub za pośred-

nictwem działu obsługi czytelników. Koszt zakupu 8051 w chwili obecnej waha się w granicach 2,00...4,00 nowych złotych, nie jest to więc dużo jak na kieszeń nawet nie zarabiającego amatora. W chwili obecnej na rynku znajduje się wersja mikroprocesora wykonana w technologii CMOS oznaczona jako 80C51.

Wszystkie parametry charakterystyczne (prądowe i napięciowe) podane w artykule będą odnosić się do tej wersji, aczkolwiek dla uproszczenia będziemy posługiwać się określeniem bez litery "C" mówiąc o typ układzie.

Mikrokontroler 8051 umieszczony jest w 40-nóżkowej obudowie (przeważnie plastikowej) typu DIL (skrót od "Dual In-Line Package", co po angielsku znaczy "obudowa dwurzędowa"). Sama ilość końcówek nie jest przerażająca, wszakże znamy inne układy np. ICL7106, które także umieszczone są w takiej obudowie.

W tym miejscu ktoś może powiedzieć: "No tak, ale opis typowej ICL ki (7106) można znaleźć prawie w każdym czasopiśmie lub podręczniku, znaczenie 40-tu wyprowadzeń też, a tu mam taki mikroprocesor, każdy mi mówi że to układ uniwersalny, a ja i tak nie wiem co mam z nim zrobić...".

Zapoznanie się z mikrokontrolerem rozpoczniemy od ogólnego poznania jego 40 wyprowadzeń, w końcu tylko to "wystaje" z obudowy i jest widoczne.

1. Końcówki o numerach 1...8 (port P1)

Są to wyprowadzenia 8-bitowego, uniwersalnego portu mikroprocesora oznaczonego w literaturze jako P1 (port nr 1 jak kto woli). Jeżeli słowo "port" nie

jest do końca jasne, posłużę się porównaniem (choć mało dokładnym) do układu typu rejestrowego o 8-miu wyprowadzeniach (8-bitach - stąd nazwa 8-bitowy). Czy pamiętamy układ serii TTL - 74198? Jest to coś w tym stylu, tylko że bardziej uniwersalne. Dla tych, którzy nie wiedzą, co to 74198, inne proste porównanie. Port, jak każdy port, pomaga w przyjmowaniu i wysyłaniu, tyle że nie towarów, lecz informacji.

Port może pełnić rolę wyjścia informacji binarnej (czyli że procesor może ustawiać stany logiczne na końcówkach tego portu). Tak więc, jeżeli zachodzi potrzeba, procesor może np. wpisać do portu P1 dowolną liczbę binarną z zakresu 0...255, np. 48. Binarnie liczba 48 = 00110000₂. Oznaczenia poszczególnych końcówek portu P1 wskazują na kolejną pozycję bitu (cyfry liczby binarnej), co pokazuje rysunek 4.

Tak więc końcówka P1.7 (najstarsza) przyjmie poziom logiczny 0, końcówka P1.6 poziom 0, P1.5 poziom 1 itd.

Ale co daje "zapisanie jakiejś liczby do portu P1"? Otóż zastosowań może być wiele. Najprostsze z nich obrazuje **rysunek 5**. Do każdego wyprowadzenia portu P1 dołączono układ z przełącznikiem, którego styki załączają

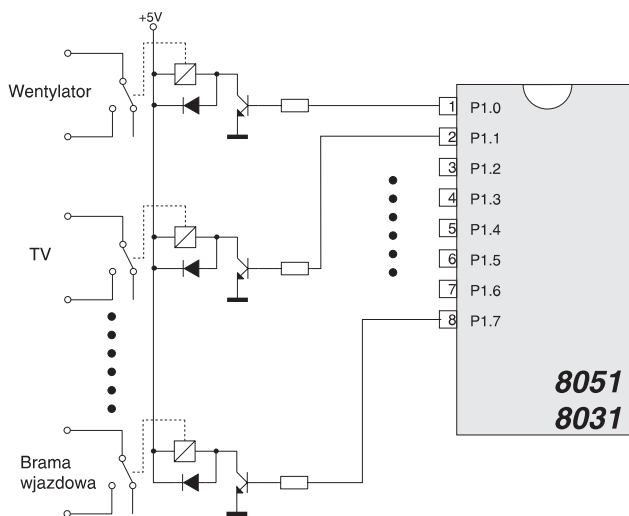
dowolne urządzenie elektryczne np. w mieszkaniu. W sumie na rysunku jest ich osiem,

lecz w praktyce nie musimy korzystać ze wszystkich wyprowadzeń portu. Przy takim wykorzystaniu portu program zawarty w mikroprocesorze może na przykład włączać i wyłączać oświetlenie w mieszkaniu. Uzyskamy świetny symulator obecności domowników.

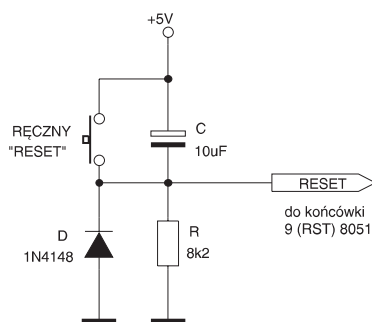
Projektując układ wykonawczy należy mieć na uwadze maksymalną obciążalność każdego z wyprowadzeń portu P1, z reguły wynosi ona 10mA (w obecnie oferowanych wersjach procesora) na każdy pin. Można zatemysterować za pomocą portu maksymalnie do czterech wejść TTL serii standard.

Istotną zaletą portów uniwersalnych procesora (w tym także P1) jest możliwość indywidualnego ustawiania poziomu logicznego na każdym wyprowadzeniu niezależnie. Nie trzeba zatem zapisywać całej liczby do portu aby np. zmienić stan tylko na jednym wyprowadzeniu, wystarczy ustawić (rozkazem zwanym SETB) lub wyzerować (rozkazem CLR) odpowiedni bit rejestru portu P1, toteż np. ustawienie pinu P1.5 na logiczne "0" nastąpi poprzez wydanie polecenia: CLR P1.5 ("clr" - clear, ang. zeruj, wyczyść). Niechcący zahaczyliśmy o programowanie, ale o tym będziemy mówić szczegółowo przy innej okazji.

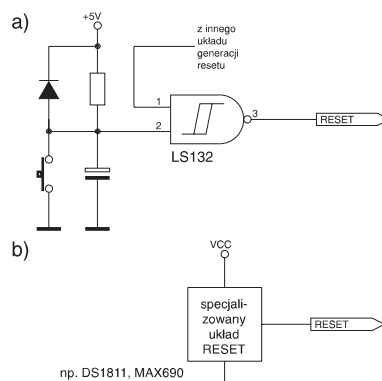
Końcówki dowolnego portu procesora mogą pełnić zarówno rolę wejść, jak wyjść.



Rys. 5. Najprostsze wykorzystanie portu P1.



Rys. 6. Podstawowy (użytkowy) układ resetowania 8051.



Rys. 7. Inne rozwiązania układów RESET.

Port (cały lub niektóre z jego pinów), podobnie jak przy zapisie, można ustawić także jako wejście informacji logicznej. Każde z wyprowadzeń staje się wtedy wyjściem o wysokiej impedancji, dzięki temu dowolny poziom logiczny podany z wyjścia jakiegoś układu cyfrowego (np. z wyjścia bramki układu TTL lub CMOS) może być odczytany poprzez piny portu a informacja czy tym stanem była logiczna "1" czy "0", zostaje wykorzystana przez procesor dla dalszego jego działania w zależności od spełnianej akurat funkcji. Krótko mówiąc, procesor może odczytać stany logiczne, jakie z zewnątrz podano na końcówki portu.

Oczywiście poziomy logiczne napięcie wejściowych portu P1 (oraz każdego innego) muszą zawierać się w przedziale napięć zasilania mikrokontrolera, czyli w zakresie 0...5V. Detekcja poziomów logicznych odbywa się jak dla bramek CMOS, stąd wartości progowe napięć tych stanów są zbliżone do połowy napięcia zasilającego. Istotną informacją jest fakt że w trybie "odczytu" z portu P1 końcówki są wewnętrznie podłączone (podciągane) do plusa zasilania poprzez wbudowane w 80C51 rezystory, co wymusza odczyt wysoki z portu w wypadku niepodłączenia końcówki portu.

2. Końcówka 9 (RST)

Z tematem mikroprocesorów czy mikrokontrolerów nierozłącznie wiąże się pojęcie "resetowania", czy jak kto woli "kasowania" układu. Czynność ta wykonywana poprzez podanie logicznej "1" na te wyprowadzenie na pewien okres czasu (jaki - o tym później) powoduje skasowanie układu, a więc natychmiastowe przerwanie wykonywanych czynności i rozpoczęcie cyklu działania procesora od samego początku (tak jakbyśmy włączyli zasilanie układu).

Czas trwania dodatniego impulsu kasującego zależy od częstotliwości z jaką

pracuje mikroprocesor. Wyjaśnię to dokładniej w dalszej części artykułu. Z reguły w typowych zastosowaniach czas 1ms w zupełności wystarcza.

W układach praktycznych do końcówki RST dołącza się mniej lub bardziej skomplikowany układ który generuje wymagany impuls zerujący najczęściej w trzech przypadkach:

- po włączeniu zasilania układu
- na nasze żądanie - poprzez np. przyciśnięcie klawisza (umieszczonego z reguły na płytce drukowanej tuż obok procesora).
- w sytuacjach awaryjnych, kiedy np. poprzez zakłócenie najczęściej na liniach zasilających nastąpi błędne działanie układu mikroprocesora, w żargonie często określa się to mianem "zawieszenia" lub "niekontrolowanej pracy" układu.

Trzeci przypadek dotyczy bardziej złożonych układów stosowanych szczególnie w automatyce i elektronice przemysłowej. My najczęściej spotkamy się z dwiema pierwszymi sytuacjami. Przykładowy układ zapewniający prawidłowy skasowanie i ponowny start procesora 80C51 przedstawia rysunek 6. Głównym

elementem układu "resetu" jest kondensator elektrolityczny C. Z reguły jego wartość powinna wynosić 10...22μF. Jest on niezbędny do prawidłowej generacji impulsu resetu przez układy wewnętrzne mikrokontrolera.

W starszych wersjach 8051 wykonanych w technologii HMOS, niezbędny okazał się dodatkowy rezystor blokujący wejście RST do masy, co zapobiegało wymuszeniu stanu niskiego na tym pinie podczas normalnej pracy układu. W nowych katalogach opisujących układy w wersji CMOS, rezystor jest ten pomijany, aczkolwiek w praktycznych układach powinniśmy przewidzieć miejsce na płytce drukowanej, ze względu na różnorodność procesorów serii 8051.

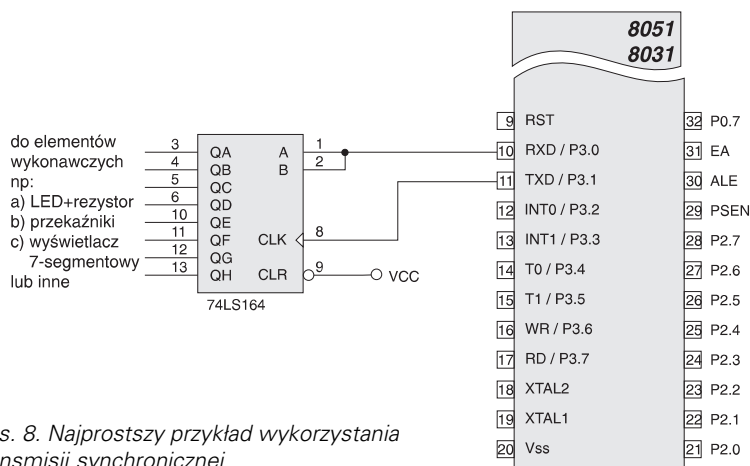
Niech za przykład posłuży fakt, że producent najnowszych procesorów z rodziny 8051 w kartach katalogowych najnowszych wersji z wewnętrzną pamięcią EEPROM typu "Flash" zaleca stosowanie tego rezystora pomimo iż produkowane układy są wykonane w wersji CMOS. My możemy stosować rezystor o wartości 8,2...10kΩ.

Widoczny na rysunku 6 klawisz służy do resetowania procesora bez konieczności wyłączania napięcia zasilającego. Toteż w każdej chwili użytkownik może przerwać wykonywanie programu przez procesor.

Na rysunku 7 pokazano inne, bardziej złożone wersje układów pełniących funkcję resetu, lecz w naszym przypadku w zupełności wystarczy wersja z rysunku 6.

3. Końcówki o numerach 10...17 (port P3)

Podobnie jak w przypadku portu P1, port P3 może pełnić wszystkie opisane wcześniej funkcje - może być wyjściem lub wejściem. Dodatkowe symbole na rysunku 3 tuż obok wyprowadzeń portu P3 sugerują że port może też spełniać inne dodatkowe funkcje. I tak też jest.



Rys. 8. Najprostszy przykład wykorzystania transmisji synchronicznej.

Też to potrafisz

Piny P3.0 (RXD) i P3.1 (TXD) mogą pełnić rolę portu transmisji szeregowej. W praktyce poprzez te dwa wyprowadzenia można przesyłać informacje (bajty i bity) z i do procesora z innych układów cyfrowych w sposób szeregowy, tzn. bit po bicie.

Ciekawostką niech będzie też fakt, że przesyłanie to może odbywać się na kilka sposobów:

- synchronicznie - wtedy pin P3.0 pełni rolę dwukierunkowej magistrali szeregowej, po której przesyłane są dane, zaś pin P3.1 generuje sygnał taktujący, pełniąc rolę zegara (podobnie jak w szeregowych rejestrach przesuwanych np. 74164, 74165). **Rysunek 8** obrazuje sposób transmisji synchronicznej do zewnętrznego 8-bitowego rejestru TTL typu 74164.

- asynchronicznie - kiedy z góry zadajemy prędkość transmisji pomiędzy naszym procesorem 8051 a innym, zewnętrznym układem np. łączem RS232c komputera PC. W takim przypadku końcówka P3.0 - RXD pełni rolę odbiornika przesyłanych szeregowo danych (pierwsza litera symbolu "R" oznacza receive - ang. odbiór), zaś końcówka P3.1 - TXD nadajnika ("T" - transmit - ang. nadawanie).

Ponadto rozróżnia się kilka trybów pracy asynchronicznej.

Tych, którzy nie zrozumieli dokładnie dodatkowych funkcji wyprowadzeń RXD i TXD pocieszam, że temat ten wyjaśnię dokładnie w rozdziale na temat sposobów komunikacji szeregowej w jednym z kolejnych odcinków cyklu.

Alternatywną funkcją końcówek P3.2 (INT0) oraz P3.3 (INT1) jest funkcja detekcji przerwań zewnętrznych. Dla tych czytelników, którzy nie wiedzą, co to oznacza, wyjaśniam, że pojęcie przerwania w tym przypadku odnosi się do zmiany stanu logicznego (na omawianym wyprowadzeniu P3.2 lub P3.3) z "1" na "0". W efekcie "we wnętrzu" procesora 8051 zostaje ustawiona tak zwana flaga (nazywana także jako "znacznik zgłoszenia przerwania", co w odniesieniu do techniki cyfrowej można wyobrazić sobie jako przerzutnik). Konsekwencją tego jest automatyczne przerwanie wykonywania przez procesor programu i natychmiastowe przejście do wykonania czynności ściśle określonych przez programistę. Ciąg takich czynności nazywany jest w technice mikroprocesorowej: "procedurą obsługi przerwania". Najprostszą analogią do zasady działania dowolnego przerwania (także zewnętrznego typu INT0 lub INT1) jest np. sytuacja, kiedy sprzątam mieszkanie, czyli wykonujemy określone czynności, powiedzmy odkurzanie. Po tym mamy za zadanie sprzątnąć kurz z pólek, a następnie umyć okna. W pewnej chwili rozlega się

gwizdek czajnika, więc oczywiście przerywamy wykonywanie - tu użyję sformułowania: "pętli głównej programu", którą jest sprzątanie pokoju - i szybko biegniemy wyłączyć gaz. Wykonaliśmy dwie dodatkowe czynności: biegniemy do kuchni i wyłączyliśmy czajnik, czyli można powiedzieć, że wykonaliśmy "procedurę obsługi przerwania" (wyłączenia czajnika, jak kto woli). Wykrycie zmiany stanu logicznego na końcówkach przerwań zewnętrznych INT0 i INT1 wiąże się ze spełnieniem jednego warunku, a mianowicie, aby czas od wspomnianego ujemnego zbocza sygnału zgłoszenia przerwania do ponownego przejścia w stan wysoki był odpowiednio długi. Podobnie jak w przypadku warunku sygnału RST, czas ten zależy od częstotliwości zegara mikroprocesora.

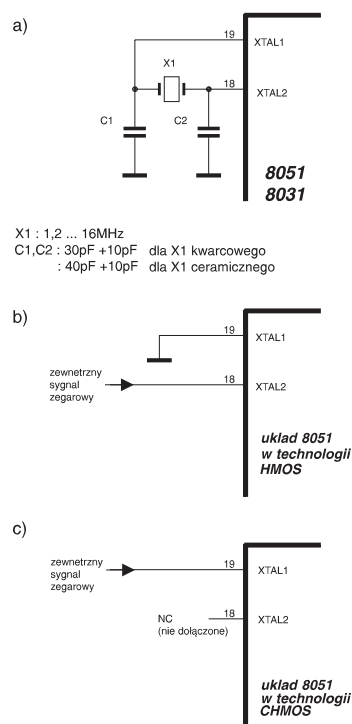
Dokładnie sposób działania systemu przerwań procesora 8051 przedstawię w kolejnych odcinkach.

Końcówki (P3.4 i P3.5) oznaczone na diagramie z rysunku 3 jako T0 i T1 pełnią dodatkową funkcję wejść uniwersalnych, programowalnych liczników, wbudowanych w strukturę 8051. Procesor 80C51 zawiera dwa bliźniacze liczniki T0 i T1 (oznaczenie takie same jak końcówki). Maksymalnie mogą one zliczać do $2^{16} = 65536$, po czym zostają wyzerowane. Liczniki te oprócz zliczania impulsów z wejść T0 i T1 mogą także zliczać impulsy wewnętrzne, pochodzące z generatora mikrokontrolera. W praktyce wykorzystywane jest to np. do odmierzenia określonych odcinków czasu np. przy funkcji zegarka. Jak wspomniałem wcześniej, liczniki mogą być programowane przez użytkownika, a więc można np. zmniejszyć ich pojemność (do 2^8 lub 2^{13}), można także zapisać w nich wartość początkową, zatrzymać je w dowolnym momencie lub uruchomić. Szczegóły w rozdziale na temat układów licznikowych procesora 8051.

Pozostały do omówienia wyprowadzenia P3.6 i P3.7, oznaczone jako WR \backslash i RD \backslash .

Jak pisaliśmy we wcześniejszych numerach EdW, prawie każdy mikrokontroler posiada możliwość współpracy z pamięcią zewnętrzną, którą przecież trzeba zaadresować. W pamięci tej można przechowywać istotne z punktu widzenia użytkownika dane, np. poziom temperatury z ostatnich dni półrocza (jeżeli procesor pracuje w układzie stacji meteorologicznej), lub inne w zależności od potrzeb.

Aby zapisać takie informacje w zewnętrznej pamięci danych potrzebne są oprócz podania adresu komórki pamięci do której ma nastąpić zapis, także sygnały sterujące zapisem lub w przypadku odczytywania - odczytem z pamięci.



Rys. 9. Typowe układy zewnętrznego oscylatora kwarcowego.

Właśnie pin WR \backslash jest sygnałem zapisu do zewnętrznej pamięci danych, a końcówka RD \backslash wysyła sygnał do odczytu. W praktycznych zastosowaniach jako elementy pamięci wykorzystuje się układy statycznych RAM - czyli w skrócie SRAM.

Procesor 8051 potrafi zaadresować maksymalnie 65536 (2^{16}) komórek pamięci (bajtów), ale o tym później.

I to tyle na temat alternatywnych funkcji portu P3, nie zapominajmy jednak że port P3 (lub niektóre z jego pinów) może pełnić rolę zwykłego, uniwersalnego portu wejścia- wyjścia, podobnie jak P1.

4. Końcówki 18 i 19 (XTAL1 i XTAL2)

Końcówki te służą do dołączenia zewnętrznego rezonatora kwarcowego o częstotliwości zależnej od potrzeb użytkownika, ale także od wersji układu 8051.

W praktyce częstotliwość ta może wynosić od 1,2MHz do 12...16MHz, na rynku spotyka się także wersje procesorów pracujące przy wyższych częstotliwościach nawet do 40MHz, a także przy niskich - nawet do pojedynczych herców w wypadku procesorów 80C51 w wersji statycznej (np. 89C51 firmy Atmel).

Dołączony do tych pinów rezonator kwarcowy po uzupełnieniu o dodatkowe kondensatory o wartości z reguły z przedziału 22...40pF (w zależności od wartości rezonatora), umożliwia pracę wbudo-

wanemu w 8051 generatorowi, który "napędza" cały mikroprocesor. Oczywiście od częstotliwości rezonatora ściśle zależy szybkość działania naszego mikrokontrolera. Typowy układ zewnętrznego oscylatora przedstawia rysunek 9a.

Częstotliwość, z jaką pracują wewnętrzne układy mikroprocesora, jest określona wzorem:

$$F = f_{\text{xtal}}/12,$$

gdzie f_{xtal} jest częstotliwością rezonatora kwarcowego.

Powodem takiego podziału częstotliwości rezonatora jest wewnętrzna architektura wszystkich procesorów serii 8051. Wiąże się z tym pojęcie "cykli maszynowych procesora" o których znaczeniu napiszę w rozdziale na temat oscylatora 8051.

W każdym razie z praktycznego punktu widzenia, przedstawiony na rysunku 9 układ, podobnie jak układ resetu z rys. 6 jest niejako obowiązkowym (przynajmniej na etapie nauki programowania).

Końcówka XTAL1 (pin 19) w układach w wersji CMOS może także pełnić rolę wejścia zewnętrznego sygnału zegarowego o częstotliwości w zakresie, jak opisano w przypadku stosowania rezonatora kwarcowego. Wtedy rezonator i dodatkowe kondensatory są zbędne.

W przypadku gdy mamy do czynienia z wersją w technologii HMOS wejściem takiego sygnału jest XTAL2 (pin 18). W obu przypadkach pozostały pin powinien być nie podłączony. Dokładnie sytuację tę wyjaśnia rys. 9b i c.

5. Końcówka 20 (Vss)

Podobnie jak w większości układów cyfrowych ostatnie wyprowadzenie w "dolnym rzędzie" obudowy jest końcówką ujemnego napięcia zasilającego - masy (GND). W przypadku układów CMOS podaje się oznaczenie Vss co

oznacza biegun ujemny napięcia zasilającego. W naszych zastosowaniach będziemy dołączać ten pin do masy przysługującego układu elektronicznego.

6. Końcówki o numerach 21...28 (port P2)

Są to wyprowadzenia drugiego 8-bitowego portu procesora. Port P2 spełnia wszystkie funkcje podobnie jak P1. Dodatkowo poprzez końcówki portu P2 podawana jest w razie potrzeby starsza część adresu (A8...A15) przy dostępie do zewnętrznej pamięci danych (SRAM) a także programu (np. EPROM). Sposób w jaki to się odbywa opiszemy przy okazji "dołączania pamięci zewnętrznej do mikrokontrolera 8051".

7. Końcówka 29 (PSEN)

W przypadku pracy procesora z zewnętrzną pamięcią programu (np. EPROM) końcówka ta wysyła sygnał odczytu z tej pamięci. W praktyce jest ona dołączona do wejścia OE współpracującej pamięci EPROM. Procesor chcąc odczytać kolejny rozkaz (polecenie do wykonania) z zewnętrznej pamięci programu podaje poziom niski na końcówkę "PSEN" a następnie dokonuje odczytu.

Dzieje się tak w ściśle określonych warunkach, synchronicznie z częstotliwością zegara procesora. Jeżeli posiadamy wersję procesora z wewnętrzną pamięcią (typu EPROM lub EEPROM), i wykorzystujemy pracę z tą wewnętrzną pamięcią, końcówka 29 powinna pozostać niepodłączona.

8. Końcówka 30 (ALE)

O końcówce 30 powiemy przy okazji omawiania portu P0.

9. Końcówka 31 (EA)

Powinna być dołączona do masy, jeżeli mikroprocesor pobiera rozkazy z zewnętrznej pamięci programu (patrz

pkt.7), lub do plusa zasilania (+5V) jeżeli z wewnętrznej.

W pewnych układach procesor pomimo że posiada wewnętrzną pamięć programu, ze względu na zbyt małą jej pojemność, musi sięgać do zewnętrznej pamięci. W takim przypadku pin EA\ powinien być dołączony do plusa zasilania, tak aby procesor po jego "resecie" mógł rozpocząć pracę pobierając rozkazy z wbudowanej pamięci programu. Należy także pamiętać że dołączenie EA\ do masy blokuje wewnętrzną pamięć programu jeżeli ona istnieje.

W praktyce jest to często stosowany chwyt, kiedy kupujemy w sklepie na ogół kilkakrotnie tańszą wersję procesora z pamięcią wewnętrzną typu ROM. W pamięci takiej najczęściej zapisany jest pewien program lecz, z naszego punktu widzenia jest on zupełnie bezużyteczny. Toteż aby w pełni wykorzystać walory mikroprocesora (oczywiście przy pracy z zewnętrzną pamięcią programu) bez uruchamiania nieznanego nam programu, blokujemy pamięć ROM poprzez zwarcie EA\ do masy.

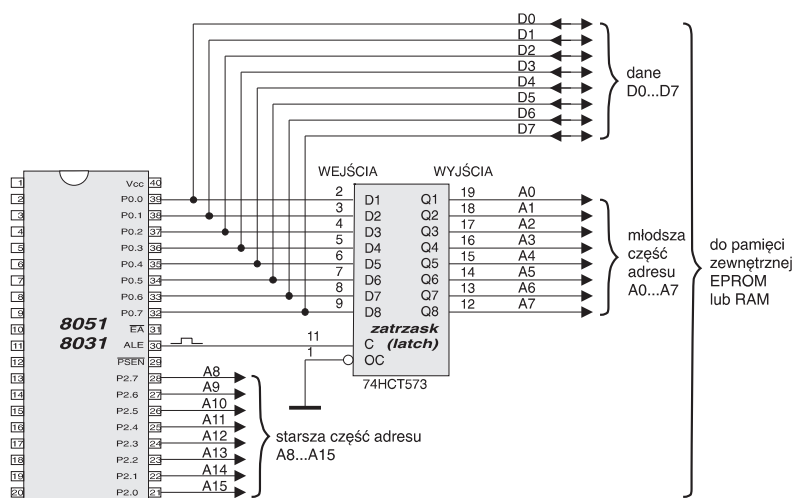
10. Końcówki o numerach 32...39 (port P0)

Trzecim i ostatnim uniwersalnym portem procesora 8051 jest P0. Podstawowe funkcje portu jako dwukierunkowej bramy do wymiany danych są takie same jak w przypadku portów P1 i P2. zasadniczą różnicą jest jednak zwiększona obciążalność (do 8 wejść TTL) tego portu oraz fakt nie posiadania wbudowanych rezystorów podciągających końcówki portu do plusa zasilania w wypadku odczytu.

Dlatego przy projektowaniu dowolnych układów wyjściowych dołączanych do tego portu należy uwzględnić wspomniane właściwości tak, aby np. odpowiednio spolaryzować bazy tranzystorów z rys. 5.

Drugą bardzo ważną rolą, jaką pełni P0, jest funkcja multiplexowanej magistrali danych (8-bitów: D7...D0) i młodszej części adresu (A7...A0). Multiplexowanej w praktyce znaczy "przełączalnej", czyli raz na końcówkach portu P0 procesor może wystawić bajt danych (np. do zapisu do zewnętrznej pamięci danych), w innym przypadku adres, w celu wybrania potrzebnej komórki z pamięci SRAM, do której ma być zapisana.

Bardziej wnikliwy czytelnik zauważy, że przecież do zapisu danej w zewnętrznej pamięci SRAM potrzeba w sumie 16 sygnałów adresu (A0...A15) oraz 8-bitów (sygnałów) danej. Potrzebne są zatem 3 ośmiobitowe porty (2 na adres i jeden na daną), a my mamy do dyspozycji tylko dwa P2 i P0.



Rys. 10. Dołączanie zatrasku do procesora.

Tabela 2.

| Parametr | Symbol | Wartość dopuszczalna | Wartość zalecana | Uwagi |
|---|-------------|----------------------|------------------|---|
| napięcie zasilania | Ucc-Uss | 6,6V | 5V ± 20% | |
| napięcie wzgl. masy na dolnej końcówce układu | -1,0...7,0V | | | w zakresie napięcia zasilania |
| moc rozpraszana | Ptot | 1W | nie dotyczy | |
| temperatura pracy | | 0°C...70°C | 0°C...70°C | dla wersji specjalnych układu zakres pracy może być większy |
| temp. przechowywania | | -65°C...+150°C | nie dotyczy | |

Tabela 3.

| Oznaczenie | Wbudowana pamięć programu | Uwagi |
|------------|---------------------------|--|
| 80C31 | bez pamięci | układ nadaje się do pracy po dołączeniu zewnętrznej pamięci EPROM wraz z niezbędnym zatraskiem (np. 74373/573) |
| 80C51 | 4kB ROM | wersja, która wymaga zablokowania pamięci, patrz opis pkt. 9 (reszta jak dla 80C31) |
| 87C51 | 4kB EPROM lub EPROM OTP* | procesor z wbudowaną pamięcią typu EPROM i możliwością kasowania jej promieniami UV poprzez okienko kwarcowe |
| 89C51 | 4kB EEPROM (Flash) | podobnie jak 87C51 z tym że pamięć programu można skasować drogą elektryczną przez podanie impulsu - dlatego mówi się o pamięci typu "Flash" (ang. błysk). |

* OTP (One Time Programmable) - pamięć EPROM zapisywalna jednokrotnie (kostka nie ma okienka kwarcowego umożliwiającego skasowanie zawartości pamięci na pomocą promieniowania ultrafioletowego).

I tu właśnie leży zasada multipleksowania (naprzemiennego wystawiania adresu lub danej) procesora 8051. Otóż sygnał - informacja o tym, co aktualnie znajduje się na szynie portu P0, pojawia się na wyprowadzeniu 30 oznaczonym jako ALE. Sygnał ten można nazwać "sygnałem zapisu adresu" do dodatkowego zewnętrznego układu cyfrowego. Układ ten jest 8-krotnym zatraskiem aktywowanym wysokim poziomem logicznym. W serii TTL znajdują się dwie kostki spełniające rolę układu zatraskiwania młodszej części adresu przez 8051, są to 74373 lub 74573.

Różnica między nimi polega jedynie na innym wyprowadzeniu końcówek, reszta działa tak samo. **Rysunek 10** pokazuje sposób dołączenia zatrasku do procesora 8051. W momencie kiedy 8051 wystawi na port P0 młodszą część adresu (A7...A0), daje temu sygnał, zmieniając stan na końcówce ALE z niskiego na wysoki. W efekcie po nadejściu tym razem opadającego zbocza sygnału

ALE, dana (adres) z portu P0 zostaje zapisana w zatrasku 74373 (573). Teraz na ośmiu jego wyjściach adres będzie utrzymywany niezależnie od zmieniających się stanów w porcie P0 aż do nadejścia następnego sygnału z końcówki ALE. Skoro procesor posługując się dodatkowym układem "zapisal" na zewnątrz adres, może teraz śmiało wystawić na port P0 daną, która ma być zapisana w zewnętrznej pamięci danych. Oczywiście można też odczytać dane z pamięci.

Tak więc podsumowując, przeanalizowaliśmy sposób w jaki za pomocą jednego sygnału ALE procesor 8051 może niejako rozszerzyć liczbę linii adresowych z 8 do 16.

W przypadku niekorzystania z możliwości obsługi zewnętrznej pamięci tak programu (EPROM) lub danych (SRAM) końcówka ALE (30) jest nieodłączona.

W odcinku poświęconym rozbudowie systemu opartego na '51-ce powrócimy do tego tematu, na razie istotne są informacje ogólne.

11. Końcówka 40 (Vcc)

Oczywiście jest to końcówka zasilania mikroprocesora 8051. Napięcie względem końcówki Vss (czyli masy) z reguły nie może przekroczyć 6,5V. Dlatego układ mikrokontrolera należy zasilac napięciem 5V ± 0,25V używając do tego celu dowolnego zasilacza stabilizowanego najlepiej przy pomocy znanego układu 7805.

Zasadą przy projektowaniu układów z 8051 jest blokowanie tego wyprowadzenia kondensatorem o wartości 100nF do masy układu cyfrowego. Praktycznie na płycie drukowanej należy zawsze przewidzieć miejsce na taki kondensator umieszczając go jak najbliżej samego układu procesora lub po prostu przylutowując go od strony wyprowadzeń na płytce drukowanej.

W tabeli 2 przedstawiono parametry dopuszczalne oraz zalecane przez producentów procesora 80C51 oraz pochodnych produkowanych w wersjach CMOS.

Na koniec pozostaje jeszcze krótkie wyjaśnienie oznaczenia samego kontrolera i kryjących się w nim dodatkowych istotnych dla nas informacji. Problem ten dokładniej przedstawia tabela 3.

Sławomir Surowiński

W poprzednim odcinku zapoznałeś się, drogi Czytelniku ze znaczeniem poszczególnych wyprowadzeń mikrokontrolera 8051. Ze względu na chęć czysto praktycznej nauki „użytkowania” tego układu, nie opisywałem dokładnie wszystkich funkcji każdej z „nózek”, a jedynie krótko zaznajomiłem Cię z przedstawionym w EdW 5/97 tematem. Być może nie wszystkie pojęcia są dla Ciebie od razu oczywiste, lecz nie powinieneś się tym przejmować, na tym etapie poznawania mikroprocesora wszystko co Ci potrzeba to „osłuchanie się” z typowymi hasłami na temat naszego bohatera. Na późniejszym etapie – praktycznej nauki z wykorzystaniem układu elektronicznego, wiedza ta przyda Ci się z pewnością, szczególnie że wtedy zaczniemy wspólnie wchodzić w temat 8051 coraz głębiej.

W tym odcinku kolejna porcja podstawowych informacji które pozwolą Ci na oswojenie się z naszym 8051!

Pamięć mikroprocesowa

Jak zapewne pamiętasz, w pierwszym odcinku naszego cyklu mówiąc o budowie „rasowego” mikrokontrolera jednouladkowego, wspominaliśmy o drugim ważnym elemencie jego architektury, a mianowicie – pamięci. Ten odcinek naszego cyklu zostanie poświęcony właśnie jej.

Prawdopodobnie spotkałeś się wcześniej z pojęciami pamięci RAM, ROM, EPROM, EEPROM itd. Wszystkie one odnoszą się do cyfrowych układów scalonych w strukturze których możliwe jest zapisanie i przechowanie informacji. Od wielu lat na rynku elektronicznym znajduje się wiele takich układów, różniących się typem, pojemnością pamięci, technologią wykonania, wszystko to bardzo często narzuca sposób ich wykorzystywania w konkretnych rozwiązaniach układowych.

Zacznijmy od krótkiej powtórki dotyczącej samych pamięci i sposobu przechowywania w nich informacji, i tak:

- podstawową jednostką przechowywania informacji w cyfrowych pamięciach jest bit;
- bit może przyjmować jedną z dwóch wartości: logiczne 0 lub 1;
- podobnie jak np. w układzie metrycznym, gdzie w celu uproszczenia pomiarów wprowadzono jednostki pochodne odległości (metr = 100cm = 1000mm), tak w przypadku jednostek informacji wprowadzono bajt, który jest równy 8 bitom.
- w odróżnieniu do typowych dziesiętnych systemów liczenia, przy omawianiu rozmiarów jak i odwoływania się do pamięci – stosuje się szesnastkowy (heksadecymalny) zapis liczb;

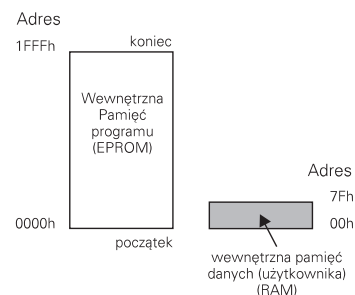


- wszystkie interesujące nas pamięci będą miały architekturę 8-bitową – czyli bajtową bowiem nasz bohater – 8051 jest mikrokomputerem 8-bitowym.
- podczas omawiania mikroprocesora, często przy okazji pamięci zamiast słowa „komórka pamięci” będziemy używać zamiennie słowa „rejestr”. Praktycznie każdy rejestr w 8051 możemy traktować jako oddzielną, posiadającą swoje miejsce (adres) komórkę w przestrzeni jego pamięci danych.
- każdy rejestr w 8051 jest 8-bitowy, niektóre z nich tworzą pary, dlatego czasami będziemy mówić o 16-bitowych rejestrach (2x8bitów = 16bitów = 2 bajty);
- powinieneś wiedzieć że w prawie każdym wskazanym (zaadresowanym) rejestrze (jak w komórce pamięci) możesz zapisać dowolną liczbę 8-bitową, lub odczytać wskazany (zaadresowany) rejestr;
- traktuj więc rejestr jako miejsce zapisu lub odczytu 8 bitów (bajtu) informacji, tak jak to ma miejsce w komórce 8-bitowej pamięci (dla maniaków cyfrówki z serii TTL rada –możesz sobie wyobrazić rejestr fizycznie jako trochę zmodyfikowany np. 74198 lub 74373).

Wracajmy jednak do tematu. Z pierwszego odcinka wiesz już że 8051 posiada 2 rodzaje pamięci. Pierwsza służy do przechowywania instrukcji programu, który ma być wykonany po

włączeniu zasilania układu. W drugiej pamięci znajdują się zmienne (tak jak w równaniach matematycznych) przechowujące określone dane i wyniki obliczeń. W 8051 dodatkowo w wydzielonej części tej drugiej pamięci znajdują się także specjalne komórki zwane rejestrami. W słownictwie związanym z 8051 używa się pojęcia SFR – z angielskiego „Special Function Registers” – rejestry specjalnego przeznaczenia („funkcji specjalnych”, jak kto woli). Tego zwrotu będziemy w przyszłości używać bardzo często, warto więc abyś sobie go zapamiętał.

Rysunek 1 przedstawia poglądową mapę pamięci zawartą w mikroprocesorze 87C51 (8751). Już wiesz że ten typ '51-ki charakteryzuje się 4 kB (kilobajtami) wewnętrznej pamięci stałej do przechowywania programu typu EPROM. Do-



Rys. 1. Organizacja pamięci wewnętrznej w mikrokontrolerze 8751.

Też to potrafisz

datkowo układ ten (podobnie jak wszystkie inne '51-ki) zawiera w swojej strukturze 128 B (bajtów) pamięci danych RAM.

Tak więc masz do dyspozycji 4kB = 4096 bajtów pamięci stałej EPROM – wszystkie komórki są zawsze numerowane (adresowane) jak wspomniano wcześniej w kodzie heksadecymalnym – tworzącej przestrzeń adresową o adresach: 0 – 4095 (dziesiętnie) lub 0000h – 1FFFh (heksadecymalnie). Dalej będziemy posługiwać się tylko tym drugim sposobem zapisu.

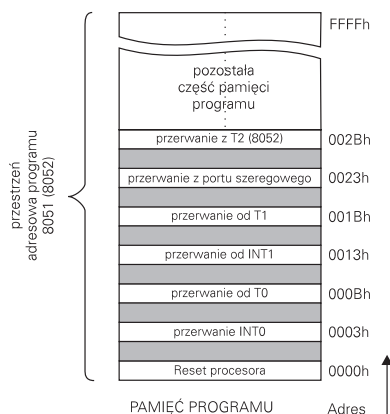
Wewnętrzna pamięć danych zajmuje adresy: 00h – 7Fh (0 – 127 dziesiętnie). Musisz w tym miejscu wiedzieć że pomimo, że adresy komórek pamięci RAM pokrywają się z częścią adresów pamięci programu, fizycznie w układzie nie występuje żaden konflikt, bowiem dostęp do omawianych obydwu rodzaj pamięci jest zupełnie inny. Mikroprocesor korzysta z innych poleceń w przypadku czytania lub zapisu do wewnętrznych 128 bajtów pamięci RAM, inne rozkazy służą do obsługi pamięci programu.

Poniżej pokrótce opiszę oba rodzaje pamięci i ich znaczenie w pracy mikrokontrolera.

Wewnętrzna pamięć programu

Program napisany przez użytkownika, dedykowany konkretnemu zastosowaniu 8051-ki powinien zostać umieszczony wewnątrz mikrokontrolera – czyli w wewnętrznej pamięci programu. Jak powiedziano wcześniej pamięć ta służy mikrokontrolerowi wyłącznie do odczytu rozkazów programu. W pamięci tej mogą być umieszczone także argumenty bezpośrednie rozkazów oraz tablice ze stałymi potrzebnymi do pewnych działań programu, np. tablica sinusów, tablica czasów zachodu słońca, lub cokolwiek innego. Mikroprocesor 8051 ma możliwość późniejszego pobrania ze swojej pamięci programu takiej stałej i wykorzystania jej np. w obliczeniach. Stała i tablice wprowadzane są przez programistę na etapie tworzenia programu, ale o tym innym razem.

Jeżeli program został przez nas utworzony a następnie zapisany w pamięci programu (o tym jak to się robi będzie mowa dalej), mikrokontroler jest gotowy do działania. Otóż po włączeniu zasilania dzięki obwodowi „Reset” (cz. I artykułu), wyzerowane zostają prawie wszystkie wewnętrzne układy mikroprocesora w tym także uwaga: „licznik rozkazów”. Ten ostatni służy mikroprocesorowi do kolejnego pobierania rozkazów z pamięci programu, a dokładnie do adresowania (czyli wskazywania) gdzie w przestrzeni adresowej pamięci programu znajduje się kolejna komenda. Jak się możesz domyś-



Rys. 2. Rozmieszczenie adresów zgłoszeń przerwań w 8051.

łać jego początkowa wartość wynosi 0 (zero), toteż pierwszym rozkazem pobranym z tej pamięci będzie ten umieszczony pod adresem 0000h.

Licznik rozkazów oznaczany jest w skrócie jako PC z angielskiego „Program Counter” – licznik programu (rozkazów) – warto o tym pamiętać. Licznik PC ma długość 16 bitów, czyli maksymalnie może liczyć do 65535 włącznie, po czym zostaje wyzerowany. Stąd wynika m.in. maksymalna wielkość pamięci programu z jakiej procesor może korzystać a mianowicie 64kB (65536 bajtów). Tak dużą pamięć posiadają niektóre mutacje '51-ki, ale prawie każdy z mikroprocesorów może współpracować z tak dużą pamięcią zewnętrzną.

W trakcie pobierania i wykonywania przez mikrokontroler kolejnych instrukcji licznik PC zmienia swoją wartość zawsze wskazując na aktualny adres kolejnego rozkazu w pamięci programu. Nasuwa się prosty wniosek, że maksymalną wartość jaką może osiągnąć licznik w naszym przypadku będzie 4095 – bowiem w naszym przykładzie z kostką 87C51 mamy do dyspozycji 4kB pamięci programu. O tym co się stanie po przekroczeniu tej wartości powiem później.

Na początek warto też wiedzieć, że oprócz wspomnianego miejsca „startowego” programu – czyli adresu 0000h (zero), w przestrzeni adresowej pamięci programu istnieje kilka innych istotnych dla programisty miejsc. Czy pamiętasz potoczne objaśnienie pojęcia „przerwanie”, pisaliśmy o tym w EdW 4/97?, jeśli nie to radzę sobie to przypomnieć. Otóż wyobraź sobie, że nasz mikroprocesor wykonuje określony program pobierając kolejne instrukcje z pamięci programu, która to jest adresowana poprzez licznik rozkazów PC. Wtem nadchodzi „przerwanie” – mikroprocesor w zależności co było jego źródłem powinien wykonać od-

powiednią dla niego procedurę obsługi (przyjęcia) przerwania.

W celu ujednolicenia systemu przerwań procesora w pamięci programu określono odpowiednie miejsca – adresy od których rozpoczyna się wykonywanie określonych procedur obsługi przerwań. W podstawowej rodzinie '51 są to adresy: 3, 11, 19, 27, 35 i 43 (03h, 0Bh, 13h, 1Bh, 23h, 2Bh szesnastkowo). Każdy z tych adresów określa początek wykonania innej procedury obsługi przerwania, dla 8051 są one następujące:

- 0003h** – przerwanie zewnętrzne z wejścia (końcówki) INT0 (pin 12)
- 000Bh** – przerw. wyniku z przepełnienia pierwszego wewnętrznego licznika T0 procesora
- 0013h** – przerwanie zewnętrzne z wejścia (końcówki) INT1 (pin 13)
- 001Bh** – przerw. wyniku z przepełnienia drugiego wewnętrznego licznika T1 procesora
- 0023h** – przerwanie wyniku z odebrania lub zakończenia wysyłania danych poprzez wewnętrzny port szeregowy mikroprocesora
- 002Bh** – przerw. wyniku z przepełnienia trzeciego wewnętrznego licznika T2.

Dodatkowo w układach 8052, 8032 (8752) występuje:

- 002Bh** – przerw. wyniku z przepełnienia trzeciego wewnętrznego licznika T2.

Na rysunku 2 zilustrowano rozmieszczenie w/w adresów zgłoszenia przerwań.

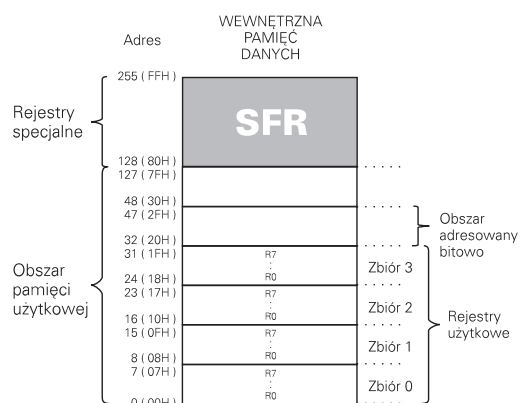
Praktycznie wygląda to tak, że w momencie zgłoszenia któregoś z wymienionych przerwań, automatycznie zachowana zostaje aktualna wartość licznika PC, a następnie zostaje wpisana do niego wartość odpowiednia do rodzaju przerwania jak opisano wyżej. Czyli np. jeżeli wewnętrzny licznik procesora T1 został przepełniony, do PC zostaje wpisana wartość 001Bh, po czym mikroprocesor rozpoczyna wykonywanie programu od tego adresu w pamięci programu. Po zakończeniu wykonywania czynności związanych z przepełnieniem T1, licznik rozkazów PC przyjmie ponownie wartość jak z przed nadejścia przerwania i program „potoczy się” dalej.

Dokładne objaśnienie działania systemu przerwań omówię przy innej okazji, na razie istotne jest abyś wiedział o istnieniu adresów specjalnych w pamięci programu procesora 8051.

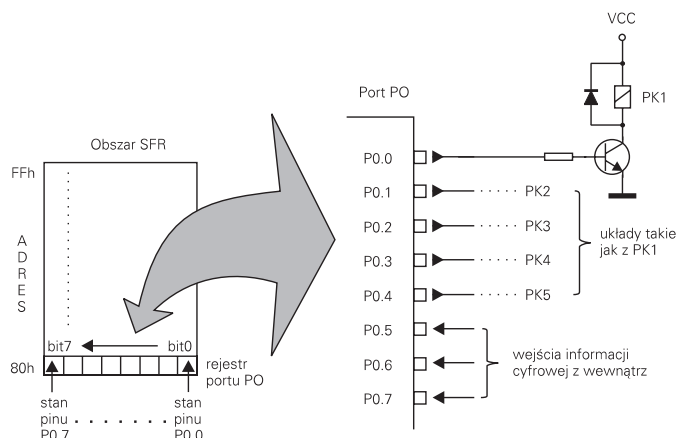
Wewnętrzna pamięć danych

W mikrokontrolerze pamięć ta przeznaczona jest dla użytkownika do przechowywania argumentów wartości zmiennych oraz wyników obliczeń arytmetyczno – logicznych.

W zależności od typu mikrokontrolera pamięć ta ma pojemność 128 lub 256 bajtów.



Rys. 3. Wewnętrzna pamięć danych w mikrokontrolerze 8051.



Rys. 4.

Dla 8051 wynosi ona 128 B (8052 – 256 B). Na rysunku 3 przedstawiono organizację wewnętrznej pamięci danych.

W przestrzeni tej pamięci można wyróżnić kilka obszarów. Dwa główne, wspomniane wcześniej to obszar pamięci użytkowej, oraz obszar rejestrów specjalnych SFR. Pamięć użytkowa zajmuje 128 komórek, adresy: 0 – 127 (00h – 7Fh), natomiast obszar SFR obejmuje adresy 128 – 255 (80h – FFh), z tym że nie wszystkie są wykorzystane przez rejestry specjalne.

I chociaż pamięć użytkownika podzielona jest na obszary, do których dostęp może odbywać się przez tzw. indeksowa-

nie obszaru, to użytkownik może adresować ją poprzez proste adresowanie.

W pamięci użytkowej komórki o adresach 0...7, 8...15, 16...23 i 24...31 tworzą cztery zbiory uniwersalnych rejestrów roboczych. Każdy z rejestrów oznacza się symbolami R0...R7. W danej chwili użytkownik ma możliwość dostępu (poprzez nazwy R0...R7) tylko do jednego „banku” (zbioru) rejestrów roboczych. Przełączanie zbiorów odbywa się poprzez odpowiednie ustawienie dwubitowego wskaźnika zwanego jako RS – z angielskiego „Register bank Swich”. O szczegółach powiemy przy okazji omawiania znaczenia poszczególnych rejestrów specjalnych SFR.

Rejestry R0 i R1 z aktywnego banku pełnią rolę wskaźników danych do pośredniego adresowania wewnętrznej pamięci danych jak i zewnętrznej. W przypadku adresowania pamięci wewnętrznej można adresować cały obszar 8051 czyli adresy 0...7Fh. Sposoby adresowania pamięci przedstawimy przy okazji „pierwszych kroków w assemblerze”.

Na rys. 3 górna część przestrzeni adresowej: 80h...FFh zajmują SFR. W tabeli 1 opisano symbole oraz nazwę każdego z nich. W tym miejscu warto zapamiętać iż rejestry specjalne stanowią niejako sprzętowy „pomost” komunikacyjny pomiędzy programistą a wszystkimi blokami funkcjonalnymi mikrokontrolera. Dla przykładu, aby „dobrać się” i odpowiednio ustawić wewnętrzny licznik T1, należy odpowiednio zmodyfikować zawartość rejestru TMOD – (rejestru trybu liczników T0 i T1) oraz TCON (rejestru sterujący licznikami oraz zgłaszaniem przerw zewnątrz INT0 i 1).

W przestrzeni adresowej SFR znajdują się także rejestry będące jednocześnie portami wejścia-wyjścia, tymi o których mówiliśmy w poprzednim odcinku artykułu. Dzięki temu możliwy jest łatwy i szybki dostęp do dowolnych bitów por-

tu czyli fizycznie do jego wyprowadzeń. Zapis do odpowiedniego rejestru portu spowoduje pojawienie się kombinacji na końcówkach mikrokontrolera, odczyt rejestru pozwoli użytkownikowi na zbadanie poziomu logicznego na wybranej linii portu.

Jak widać z tabeli nie wszystkie 128 adresów z przestrzeni SFR jest wykorzystanych. „Puste” adresy nie nadają się do wykorzystania przez użytkownika. Nie jest to bynajmniej marnotrawienie cennych bajtów pamięci, lecz czysta przezorność projektantów rodziny 8051, którzy konstruując rozszerzone wersje poczwójnej ‘51-ki wyposażają je w nowe dodatkowo bloki funkcjonalne, a w wolnych miejscach przestrzeni SFR umieszczane są dodatkowe rejestry sterujące ich pracą (wspomniane „pomosty”).

I tak np. w mikrokontrolerze 8052 umieszczono dodatkowy licznik T2, do sterowania którego niezbędne stało się zaimplementowanie w strukturze SFR rejestrów T2CON, TH2, TL2, RLDH i RLDL – patrz tabela 1.

W tym miejscu widoczny jest geniusz architektury jednoukładowców z rodziny ‘51. Otóż producenci wytwarzając nowe mutacje tych procesorów, nie muszą się martwić o kompatybilność programową, czy architekturę dostępu do poszczególnych bloków układu. W każdym przypadku dodatkowe rejestry specjalne sterujące ich pracą umieszczane są w tej samej przestrzeni SFR, w taki sam sposób dostępnej dla użytkownika. Czyli jeżeli np. któryś z producentów zechce umieścić w strukturze 8051 8-bitowy przetwornik analogowo – cyfrowy, to prawdopodobnie w wolnych miejscach obszaru SFR umieści dodatkowe rejestry: a) rejestr sterujący pracą przetwornika, oraz b) rejestr danych z przetwornika, prawda że proste, no przynajmniej z naszego punktu widzenia.

Cd. na str. 46

Tabela 1. Rejestry specjalne mikrokontrolera 8051.

| Adres | Symbol | Nazwa |
|-------|--------|---|
| E0h | ACC | Akumulator |
| F0h | B | Rejestr B |
| D0h | PSW | Słowo stanu programu |
| 81h | SP | 8-bitowy wskaźnik stosu |
| 83h | DPH | bity 8 – 15 } wskaźnik danych |
| 82h | DPL | bity 0 – 7 } DPTR |
| 80h | P0 | Port 0 |
| 90h | P1 | Port 1 |
| A0h | P2 | Port 2 |
| B0h | P3 | Port 3 |
| B8h | IP | Rejestr sterujący priorytetem przerw |
| A8h | IE | Rejestr kontrolny sterujący pracą systemu przerw |
| 88h | TCON | Rejestr kontrolny pracy liczników T0 i T1 oraz przerw INT0 i INT1 |
| 89h | TMOD | Rejestr sterujący trybem pracy liczników T0 i T1 |
| 8Ch | TH0 | bity 8 – 15 } 16-bitowy |
| 8Ah | TL0 | bity 0 – 7 } licznik T0 |
| 8Dh | TH1 | bity 8 – 15 } 16-bitowy |
| 8Bh | TL1 | bity 0 – 7 } licznik T1 |
| 88h | T2CON | rejestr sterujący licznikiem T2 (w 8052) |
| CDh | TH2 | bity 8 – 15 } 16-bitowy |
| CC | TL2 | bity 0 – 7 } licznik T2 |
| CBh | RLDH | bity 8 – 15 } Słowo ładowane |
| CAh | RLDL | bity 0 – 7 } do licznika T2 |
| 98h | SCON | Rejestr sterujący portem szeregowym |
| 99h | SBUF | Bufor portu szeregowego |
| 87h | PCON | Rejestr sterujący zasilania |

Kontynuujemy opis układów wewnętrznych mikrokontrolera 8051. W tym odcinku wyjaśnimy pojęcie i znaczenie „stosu” i jednostki arytmetyczno-logicznej

Przy okazji wszystkim „niecierpliwym” lub „wątpiącym” w swoje możliwości dotyczące programowania 8051 jeszcze raz przypominam, że w tej części cyklu nie opisujemy szczegółowo wszystkich komponentów procesora, jedynie w sposób przystępny staramy się wspólnie zrozumieć fakt istnienia tych elementów oraz ich znaczenie dla całego procesora. Autor robi to celowo, tak abyś mógł drogi Czytelniku „oswoić się” z naszym bohaterem. Wszystkie omówione elementy architektury 8051 będziemy sukcesywnie „przywoływać” z pamięci i sprawdzać ich działanie w praktyce podczas „nauki programowania 8051”. Wtedy to zdobyta i „oswojona” wiedza okaże się kluczem do sukcesu którym będzie z pewnością pierwszy napisany przez Ciebie program na 8051.



Na początek krótkie przypomnienie: w poprzednim odcinku zajmowaliśmy się wewnętrzną pamięcią programu mikroprocesora oraz wewnętrzną pamięcią danych. W jej obrębie pobieżnie omówiliśmy istnienie „rejestrów specjalnych” – SFR oraz poznaliśmy funkcję licznika rozkazów PC. Szczegółowy opis SFR oraz „sprzętowe” działanie licznika rozkazów opiszemy przy okazji omawiania cyklu rozkazowego procesora oraz wstępu do asemblera. Na tym etapie jednak naszym celem będzie poznanie pozostałych bloków funkcjonalnych 8051-ki.

Wewnętrzna pamięć danych w 8052

Wbrew pozorom tytuł tego akapitu nie wykracza poza ramy naszego minikursu na temat 8051. Jak już wiesz (z części I) mikrokontroler 8052 to rozszerzona wersja '51-ki, posiadająca:

- dodatkowy 16-bitowy uniwersalny układ licznikowy
- dodatkowe 128 bajtów wewnętrznej pamięci danych RAM.

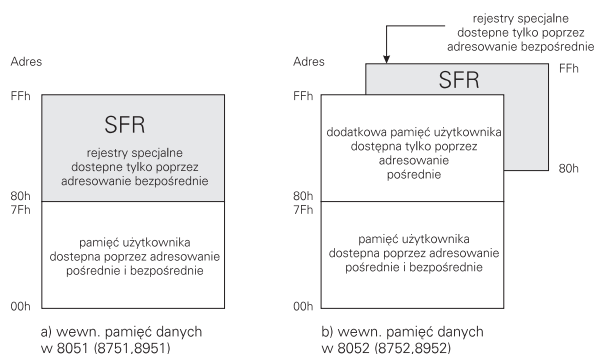
I właśnie tym drugim elementem zajmujemy się teraz. Na **rysunku 1a** przedstawiłem znaną Ci już strukturę wewnętrznej pamięci danych w 8051. Jak mówiliśmy poprzednio obszar o adresach: 00h...7Fh (128 bajtów) zajmuje pamięć danych użytkownika, którą można adresować w sposób pośredni (poprzez rejestry

wskaźnikowe R0 i R1) lub bezpośrednio odwołując się niejako „wprost” do danej komórki pamięci. Pozostałe 128 bajtów pamięci o adresach 80h...FFh to obszar omawianych w skrócie „rejestrów specjalnych – SFR”.

Obok na **rysunku 1b** pokazałem strukturę wewnętrznej pamięci danych w mikrokontrolerze 8052. Jak widać w przestrzeni tej istnieje dodatkowa, niejako zdublowana pamięć danych o adresach 80h...FFh. Przeznaczenie jej jest takie jak pamięci użytkownika o adresach 00h...7Fh, czyli w całości (128 bajtów) można ją wykorzystać do swoich potrzeb umieszczając w niej te wyniki obliczeń oraz pozostałe zmienne programowe, które po prostu „nie mieszczą” się w obrębie pamięci danych użytkownika o adresach 00h...7Fh. Ze względu jednak że ta dodatkowa, nazywana często „nakładkową”, pamięć danych pokrywa się adresowo (adresy 80h...FFh) z obszarem rejestrów specjalnych SFR, należało ją w jakiś sposób rozróżnić, tak aby np. podczas odczytu lub zapisu którejś z komórek tej pamięci nie zmodyfikować przypadkiem któregoś z rejestrów specjalnych SFR.

Otóż konstruktorzy kontrolera 8052 rozwiązali ten problem w prosty sposób umożliwiając dostęp do tej dodatkowej „nakładkowej” części pamięci, jedynie za pośrednictwem adresowania pośredniego. Toteż jeżeli np. w przyszłości, drogi

Też to potrafisz



Rys. 1. Pamięć w procesorach 8051 oraz 8052

Czytelniku, odwołasz się (zaadresujesz) do dowolnej komórki wewnętrznej pamięci danych w sposób bezpośredni a wskazywanym przez Ciebie adresem komórki będzie np. F0h (240 dziesiętnie) to z pewnością „dobierzesz” się do rejestru specjalnego o symbolu B (patrz tabela 1 w poprzednim odcinku), natomiast jeżeli wykonasz to samo tym razem jednak adresując komórkę w sposób pośredni poprzez rejestry wskaźnikowe R0 lub R1, to dokonasz zapisu (lub odczytu) komórki położonej w obszarze nakładkowym – czyli części dodatkowej pamięci danych.

Oczywiście jeżeli wykonasz tą ostatnią operację programując kostkę 8051, to w efekcie zaadresowania pośredniego komórki o adresie z zakresu 80h...FFh trafisz przysłowiową „kulą w płot”, czyli nie uzyskasz oczekiwanego efektu, bo po prostu w tym obszarze 8051 adresowanym pośrednio po prostu nie ma nic. Warto o tym pamiętać, bowiem zgodnie z zasadą kompatybilności w dół, program napisany na procesor o mniejszych możliwościach (np. 8051) z pewnością będzie pracował poprawnie na 8052, ale nie odwrotnie. To samo dotyczy każdego członka rodziny MCS-51.

Stos i wskaźnik stosu

Z pojęciem „stosu” miałeś okazję, drogi Czytelniku, spotkać się w artykule omawiającym ogólne założenia dotyczące mikroprocesorów, pisaliśmy o tym w EdW. Jeżeli nie do końca rozumiesz istotę stosu postaram się Ci ją jeszcze raz przedstawić. Otóż najprościej można stos określić jako bardzo prostą w działaniu strukturę przechowującą bajty. Pod pojęciem „przechowywania” rozumiemy oczywiście operację zapisu z następnym odczytu dowolnej zmiennej lub rejestru SFR.

Wiesz już że w przypadku takich operacji tylko z udziałem np. wewnętrznej pamięci danych użytkownika, aby dokonać zapisu (odczytu) musisz daną komórkę pamięci najpierw zaadresować – czyli po prostu podać jej fizyczny adres.

nia danych charakteryzuje właśnie stos. **Rysunek 2** wyjaśnia fizyczną budowę stosu. Jak widać wszystkie dane (bajty) przy zapisie odkładane są „na stos” jedna na drugą. Na wierzchołku stosu znajduje się zawsze ostatnio odłożona dana (w naszym przykładzie oznaczona jako X), toteż aby „dobrać się” do danej leżącej pod nią (Y) należy najpierw „zjąć” ze stosu daną X, a potem dopiero odczytać Y. Można to porównać do stosu talerzy ustawionych jeden nad drugim. Odkładamy talerze na stos i zdejmujemy ze stosu. Nie możemy wyjąć talerza „z głębi stosu” – dostajemy się do niego dopiero po zdjęciu wszystkich stojących na nim.

„Po co jednak jest ten „stos”, czy nie jest to tylko niepotrzebna komplikacja”, z pewnością wielu z Was w tej chwili zadaje sobie to pytanie. Otóż jak się okaże później a szczególnie podczas nauki programowania, struktura ta spełnia niezmiernie ważną rolę podczas wykonywania programu przez mikroprocesor. Na tym etapie powinniśmy wiedzieć tylko dwie podstawowe rzeczy: stos służy do przechowywania zmiennych lub rejestrów SFR i druga sprawa: dostęp do nich odbywa się w sposób uporządkowany – w odpowiedniej kolejności, jak opisałem wcześniej.

„No tak ale gdzie jest ten stos ?...”, już odpowiadam. W przypadku procesorów rodziny MCS-51 stos umieszczony jest... uwaga !, w wewnętrznej pamięci danych użytkownika, czyli w obszarze o adresach 00h...7Fh.

Jak wynika z rysunku 2 ilość tej pamięci zajętej przez stos będzie się zmieniać i zależeć od tego ile bajtów odłożyliśmy na ten stos.

Aby ściśle określić miejsce położenia stosu, w architekturze '51-ki znajduje się tzw. licznik stosu a fachowo

W przypadku korzystania ze stosu adresowanie jest niekonieczne. Przy takim sposobie obsługi konieczne jest jednak zachowanie odpowiedniej kolejności w zapisie i odczycie tak aby nasze cenne dane nie „pomieszały się”.

Otóż taki uporządkowany sposób przechowywania

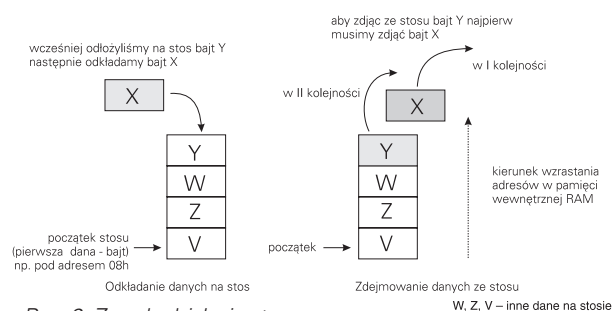
mówiąc „wskaźnik stosu”. Fizycznie jest on po prostu 8-bitowym rejestrem w obszarze SFR, położonym pod adresem 81h (patrz tabela 1 w poprzedniej części). W mnemonice (nazewnictwie) procesorów MCS-51 posiada on symbol SP z angielskiego „stack pointer” – wskaźnik stosu.

Jego zadaniem jest automatyczne wskazywanie miejsca aktualnego wierzchołka stosu. Tak więc w przypadku odłożenia bajtu na stos, wskaźnik SP jest automatycznie (bez ingerencji programisty) zwiększany o 1, w przypadku zdjęcia danej ze stosu jest on zmniejszany. Sytuację te wyjaśnia **rysunek 3**.

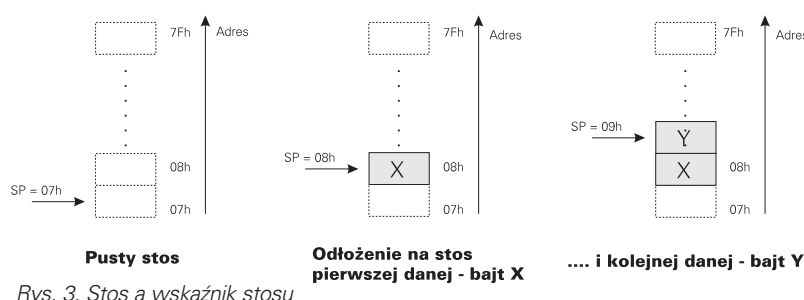
Podsumujmy więc: stos jest hierarchiczną strukturą do przechowywania danych (bajtów) z obszaru wewnętrznej pamięci RAM procesora (włączając SFR) a położenie jego wierzchołka jednoznacznie określa jego wskaźnik – SP. Przy korzystaniu ze stosu obowiązuje zasada, „ile bajtów odłożyłeś na stos, tyle potem musisz zjąć”, tak aby struktura stosu nie została zakłócona. W praktyce ma to szczególne znaczenie, bowiem stos wykorzystywany jest nie tylko poprzez świadome działanie użytkownika lecz także przechowywane są na nim ważne dla działania całego mikrokontrolera adresy powrotów z podprocedur oraz procedur obsługi przerwań, czyli innymi słowy mówiąc, aktualne zawartości 16-bitowego licznika rozkazów PC.

No tak ale przecież stos składa się z 8-bitowych komórek pamięci, a licznik rozkazów (programu PC) jest 16-bitowy. W takim przypadku procesor na stos odkłada najpierw młodszy bajt rejestru PC, a następnie starszy bajt, wskaźnik stosu SP zostaje więc zwiększony automatycznie o 2. Tak więc w prosty sposób można przechowywać inne rejestry podwójne np. wskaźnik adresu zewnętrznej pamięci – DPTR (tabela.1), składający się z dwóch 8-bitowych rejestrów DPH (adres 83h) oraz DPL (adres 82h).

W przypadku rejestru DPTR jak i innych SFR przechowywanie na stosie odbywa się „na żądanie” użytkownika – w potrzebnym dla niego momencie. O tym jak



Rys. 2. Zasada działania stosu



w praktyce i dlatego przechowuje się rejestry na stosie dowiesz się drogi Czytelniku przy okazji nauki programowania 8051.

Na **rysunku 4** przedstawiono dwie sytuacje w których używany jest stos. Pierwsza dotyczy przechowania rejestru licznika rozkazów (programu) podczas obsługi procedury po nadejściu przerwania, druga obrazuje jak świadomie można wykorzystać stos do przekazywania danych pomiędzy rejestrami. W praktyce ten ostatni przypadek jest niezmiernie rzadko wykorzystywany, lecz w tym przykładzie chodzi nam o zrozumienie samego sposobu działania struktury stosu.

Na koniec dwie pozostałe ważne informacje dotyczące stosu. Otóż po włączeniu zasilania procesora (lub jego resecie oczywiście) wskaźnik stosu SP przyjmuje domyślnie wartość 07h – czyli po prostu 7, wskazując tym samym że wierzchołek stosu – adres umieszczenia następnej danej – po odłożeniu jej na stos położony będzie w wewnętrznej pamięci danych pod adresem 08h (07h + 1 zgodnie z opisaną wcześniej zasadą).

Jeżeli więc odłożymy jakiś bajt na stos, najpierw licznik SP zostanie automatycznie zwiększony o 1 (wskazując teraz 08h), a następnie do komórki pamięci o tym adresie 08h, zostanie wpisany ten bajt. Przy zdjęciu ze stosu kolejność będzie odwrotna, najpierw zdjęty zostanie nasz bajt, a następnie zmniejszony zostanie wskaźnik SP o 1.

Wskaźnik stosu SP tak jak każdy rejestr SFR może być dowolnie modyfikowany przez programistę poprzez zapisanie w nim dowolnej 8-bitowej wartości (0...255)

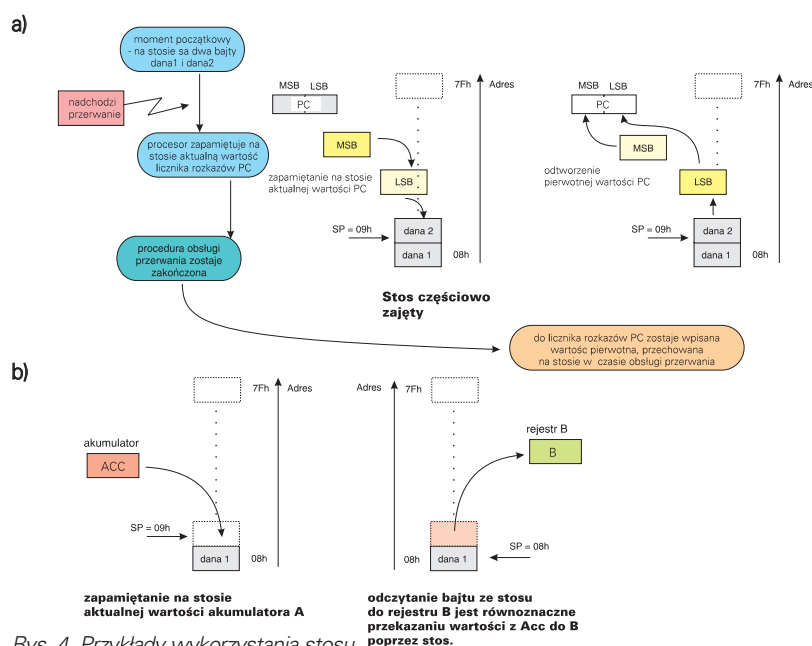
W praktyce jednak sytuacja taka występuje tylko wtedy, jeżeli chcemy zmienić położenie stosu (czyli go przesunąć) na początku wykonywania programu. Operacja ta z oczywistych względów ma sens jeżeli stos w danej chwili jest „pusty”, w przeciwnym razie przy lekkomyślnej modyfikacji wskaźnika SP wszystkie dane odłożone wcześniej na stos staną się niedostępne (przynajmniej z punktu widzenia działania samego stosu).

I tak jeżeli np. zechcesz drogi Czytelniku wykorzystywać wewnętrzną pamięć danych o adresach 08h...20h dla swoich

potrzeb (a nie na stos), musisz na początku swego programu zmodyfikować wskaźnik SP wpisując do niego wartość np. 20h, co jest jednoznaczne z powiedzeniem mikroprocesorowi: „... uważaj mikroprocesorze, twój stos będzie rozpoczynał się od adresu 21h, (a nie od 08h), adresy 08h...20h są przeznaczone dla moich potrzeb...”

Na koniec jeszcze jedna istotna uwaga. Otóż jak widać ze sposobu działania stosu, poprzez nieumiejętne korzystanie z niego bardzo łatwo jest go „zamazać” lub mówiąc inaczej zniszczyć. Przykładem niech będzie sytuacja w której:

- mikroprocesor wykonuje swój rutynowy program
- nagle nadchodzi przerwanie z wejścia INTO
- procesor przerywa działanie pętli głównej programu
- następuje skok do wykonania procedury obsługi przerwania
- zanim jednak to nastąpi procesor automatycznie zapisuje na stos aktualną zawartość licznika rozkazów PC (tak aby potem wiedzieć gdzie ma wrócić do pętli głównej programu)



f) po odłożeniu na stos licznika PC (2 bajty), procesor wykonuje procedurę obsługi przerwania

g) w tej procedurze użytkownik świadomie używa stosu do przechowywania tymczasowo pewnych wartości, zapisując na stos np. 3 bajty (te 3 bajty znajdują się w sąsiedztwie bezpośrednio „nad” bajtami z licznika rozkazów PC)

h) z powodu błędu w programie – popełnionego przez użytkownika – pod koniec procedury zostają zdjęte ze stosu dwa (a nie trzy) bajty, które przechowywały dane użytkownika

i) następuje zakończenie procedury obsługi przerwania, procesor zdejmując ze wierzchołka stosu adres powrotu do pętli głównej programu (2 bajty, które wpisuje do licznika rozkazów PC) i tu następuje „kompletny kłops” – program prawdopodobnie „zawiesi się” lub po prostu „zawaruje”.

Powód tego jest oczywisty, do licznika rozkazów PC nie zostały wpisane wcześniej przechowane 2 bajty będące pierwotną zawartością PC, a za to wpisany zostaje bajt pozostawiony przez użytkownika (zawierający najpewniej inną wartość liczbową) w procedurze obsługi przerwania, oraz starszy bajt licznika rozkazów PC.

W tej sytuacji procesor powróci w zupełnie inne miejsce programu, niż w te w którym się znajdował w momencie nadejścia przerwania, czego skutki dla działania procesora okazać się mogą opłakane.

Pamiętajmy zatem o stosie jako o ważnej strukturze w architekturze 8051, oraz o tym że tylko umiejętne i świadome, oczywiście, z niego korzysta-

Też to potrafisz

nie przynosi często efekty w postaci znacznego przyspieszenia działania programu oraz zmniejszenia jego rozmiarów. O tym jak w praktyce korzystać z dobrodziejstw stosu powiemy dowiemy się podczas nauki programowania.

Jednostka arytmetyczno-logiczna

Pod tym pojęciem kryje się jeden z elementów architektury 8051 odpowiadający za wykonywanie operacji arytmetyczno – logicznych. Blok ten nazywany w skrócie jako ALU, potrafi wykonywać operacje na liczbach (składnikach) 8-bitowych. Z matematyki wiemy że do wykonania najprostszego działania dwuskładnikowego potrzebne są: po pierwsze składniki, po drugie w wyniku działania powstaje wynik, który też należy gdzieś przechować (umieścić).

Do wprowadzenia (np. przez programistę) składników działania służą zarówno niektóre rejestry specjalne z grupy SFR jak i dowolna komórka wewnętrznej pamięci danych. Dla różnych działań występują jednak pewne ograniczenia w swobodzie umieszczania składników, lecz to temat na oddzielny artykuł.

Jednym z najważniejszych rejestrów z grupy SFR jest akumulator oznaczany dużą literą A (ang. „accumulator”).

W tabeli 1 z poprzedniej części widać że akumulator umieszczony jest pod adresem E0h (224 dziesiętnie). Rejestr ten służy jednostce ALU za miejsce pobrania argumentu oraz umieszczenia wyniku większości operacji arytmetyczno logicznych.

Rejestr ten może być adresowany bitowo (podobnie jak bajty spod adresów 20h...2Fh – patrz poprzedni odcinek), dzięki czemu możliwe jest testowanie dowolnych jego bitów bez potrzeby wykonywania dodatkowych operacji logicznych. Muszę w tym miejscu zasignalizować że dodatkowo rejestr A poza funkcjami związanymi z jednostką ALU służy do pobierania i umieszczania bajtów w zewnętrznej pamięci danych, dokładnie o tym powiemy w kolejnych odcinkach kursu.

Przy przesyłaniu tego rejestru na stos (umieszczenie lub pobranie ze stosu) wykorzystuje się adresowanie bezpośrednie tego rejestru. Wtedy opisujemy go symbolem ACC (lub Acc). Dokładnie takie przypadki poznasz przy okazji programowania 8051.

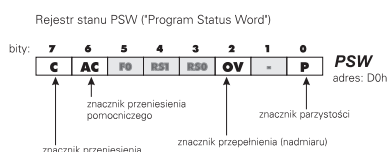
Drugim po akumulatorze ważnym rejestrem współpracującym z ALU jest, także 8-bitowy, rejestr B. Służy on do umieszczenia jednego ze składników mnożenia lub dzielenia, a po wykonaniu jednej z tych operacji w rejestrze tym umieszczany jest

– w przypadku mnożenia starszy bajt 16-bitowego wyniku mnożenia dwóch liczb 8-bitowych

– w przypadku dzielenia: reszta z dzielenia dwóch liczb 8-bitowych.

Oczywiście zarówno rejestr B jak i akumulator A mogą być wykorzystywane dowolnie jako rejestry uniwersalne.

Trzecim ważnym rejestrem związanym z ALU jest „słowo stanu programu” nazywane w skrócie jako PSW (od ang. „program status word”) – patrz **rysunek 5**. Z tabeli 1 możemy odczytać że rejestr ten wchodzi w skład SFR a jego adres to D0h (208 dziesiętnie). W skład tego rejestru wchodzi 8 bitów nazywanych znacznikami z których cztery informują o przebiegu wykonania operacji arytmetyczno – logicznych. I tak:



Rys. 5. Zawartość rejestru stanu

PSW.0 (bit 0) – oznaczany jako P, to znacznik parzystości, ustawiany automatycznie w każdym cyklu maszynowym wskazuje na to czy liczba jedynek (na poszczególnych) pozycjach bitowych w akumulatorze A jest parzysta (P=1) czy nieparzysta (P=0).

PSW.2 (bit2) – oznaczany jako OV, to znacznik przepełnienia (nadmiaru), ustawiany w wyniku wykonania dodawania lub odejmowania, a przy operacji dzielenia ustawienie go wskazuje na dzielenie przez zero.

PSW.6 (bit 6) – oznaczany jako AC, to znacznik przeniesienia pomocniczego, do którego wpisywane jest przeniesienie lub pożyczka z bitu 3, wykorzystywany jest przy korekcji dziesiętnej liczb.

PSW.7 (bit 7) – znacznik przeniesienia oznaczany jako C, do którego następuje przeniesienie z najbardziej znaczącego bitu w wyniku wykonania operacji logicznych przesunięć liczb 8-bitowych lub w wypadku przekroczenia wyniku poza zakres liczb zapisanych w naturalnym kodzie dwójkowym (>255).

Pozostałe znaczniki nie mają związku z ALU, toteż ich omówieniem zajmniemy się przy innej okazji.

W praktyce najczęściej nie jest konieczne pamiętanie o wszystkich wymienionych znacznikach, no może poza znacznikiem C (PSW.7). Jak się okaże podczas nauki programowania, znaczniki te działają jak gdyby automatycznie, to znaczy istnieją instrukcje programowania 8051, które uwzględniają wspomniane znaczniki, toteż nie jest koniecznym badanie samego bitu słowa PSW, a jedynie wykonanie odpowiedniej instrukcji która uwzględni odpowiedni stan danego znacznika.

Znając pobieżnie główne 3 rejestry związane z ALU zapoznamy się wstępnie z operacjami jakie można wykonywać przy jej pomocy na liczbach 8-bitowych, są to

a) operacje arytmetyczne

- dodawanie argumentów
- dodawanie z przeniesieniem
- odejmowanie z pożyczką

W tych trzech przypadkach pierwszy z argumentów operacji (składnik lub odjemna) umieszczana jest w akumulatorze, drugi składnik lub odjemnik umieszczony jest w wewnętrznej pamięci danych, lub jest argumentem bezpośrednim rozkazu. Wynik działania umieszczany jest w akumulatorze. Dodatkowo w słowie PSW ustawiane są odpowiednie znaczniki: przeniesienia C i nadmiaru OV, co jest sygnałem przekroczenia zakresu liczb 8-bitowych odpowiednio bez lub ze znakiem.

Pozostałe operacje arytmetyczne to:

- mnożenie dwóch 8-bitowych liczb bez znaku, gdzie jeden składnik wpisywany jest do akumulatora drugi do rejestru B, 16-bitowy wynik umieszczany jest w rejestrach B.A odpowiednio starszy bajt w B, młodszy w A;
- dzielenie dwóch liczb 8-bitowych, gdzie dzielna umieszczana jest w akumulatorze A, a dzielnik w B, 8-bitowy wynik dzielenia znajduje się po tej operacji w A, natomiast B przechowuje resztę z dzielenia.
- inkrementacja (zwiększanie o 1) lub dekrementacja (zmniejszenie o 1) akumulatora lub dowolnej komórki w wewnętrznej pamięci danych
- korekcja dziesiętna wyniku zapisanego w akumulatorze

b) operacje logiczne

- logiczna suma (OR)
- iloczyn logiczny (AND)
- różnica symetryczna (EXOR)
- negacja (NOT) zawartości akumulatora A
- przesuwanie cykliczne akumulatora w lewo lub prawo, z lub bez przeniesienia (znacznika C → PSW.7).

Ze wszystkimi operacjami tak logicznymi jak i arytmetycznymi zapoznasz się drogi Czytelniku podczas omawiania poszczególnych instrukcji, na razie ważne jest abyś zapamiętał omówione tu podstawowe zagadnienia związane z ALU procesora 8051 i pochodnych.

Nie załamuj się, jeśli coś nie jest dla Ciebie do końca jasne. Spróbuj rozjaśnić obraz materiałem z poprzednich odcinków. Jeśli to nie pomoże, zrozumiesz te szczegóły przy omawianiu praktycznych przykładów.

Sławomir Surowiński
c.d. w EdW 8/97

Uwaga! W części 2 (EdW 6/97, s. 41) zamiast: „Pamięć mikroprocesora” powinno być „Pamięć mikroprocesora”.

Kontynuujemy opis mikrokontrolera 8051. W tym odcinku powiemy w jaki sposób można dołączyć zewnętrzną pamięć programu oraz danych. Podane konkretne rozwiązania sprzętowe poparte schematami elektrycznymi, z pewnością dadzą Ci pojęcie na temat konstruowania podstawowych bloków funkcjonalnych małego systemu mikroprocesorowego. Opis takiego układu będącego jednocześnie bazą do naszej przyszłej nauki programowania 8051 prezentujemy w niniejszym numerze EdW, warto jednak abyś najpierw zapoznał się z niniejszym artykułem.



Zewnętrzna pamięć programu

Jak się dowiedziałeś z wcześniejszych części naszego cyklu, procesor 8051 i mu podobne mają możliwość dołączenia dodatkowej zewnętrznej pamięci programu, np. typu EPROM. W pamięci tej podobnie jak w wewnętrznej pamięci programu zaszytej w kostce 8751 (8752) programista umieszcza poszczególne rozkazy programu, używając do tego celu np. programatora pamięci EPROM lub programatora do programowania mikroprocesorów z rodziny MCS-51. Wiesz już że procesor może pracować w następujących konfiguracjach:

- tylko z wewnętrzną pamięcią programu (dla kostek 8751, 8752)
- z wewnętrzną i zewnętrzną pamięcią programu jednocześnie, w tym przypadku zewnętrzną pamięć programu stanowi jak gdyby „przedłużenie” pamięci wewnętrznej.
- oraz tylko z zewnętrzną pamięcią programu np. typu EPROM.

Pierwszy tryb jest bardzo wygodny, pozwala na „pełne” wykorzystanie wszystkich zalet mikrokontrolera jednokładowego w całym tego słowa znaczeniu. Programista wykorzystując procesor w wersji z wbudowaną pamięcią programu (8751, 8951, xx52) cały kod swego programu umieszcza wewnątrz kości, dzięki czemu ma do dyspozycji wszystkie porty mikrokontrolera – w tym także P0 i P2. (patrz poprzednie odcinki naszego cyklu).

Kostka 87C51 (89C51) ma „tylko” 4kB (8x52 8kB pamięci programu), najczęściej te „tylko” w zupełności wystarcza, lecz zdarza się że jest to za mało, wtedy konieczne jest dołączenie dodatkowej zewnętrznej pamięci programu w postaci kostki EPROM. W tym przypadku (jak wspominałem w poprzednim odcinku) maksymalna długość programu równa wielkości obu pamięci zewnętrznej jak i wewnętrznej nie może przekroczyć 64kB – czyli 65536 8-bitowych słów.

Czyli że np. do kostki 87C52 (zawierającej 8 kB wewnętrznej EPROM) można dołączyć maksymalnie 56kB EPROM z zewnątrz.

Trzeci tryb pracy tylko z zewnętrzną pamięcią programu jak pamiętasz z pierwszej części cyklu wymaga zwarcia wyprowadzenia /EA (pin 31 procesora 8051/52) do masy. Parę lat temu, kiedy procesory '51 z wbudowaną wewnętrzną pamięcią programu kosztowały niemało, natomiast kostki bez niej można było nabyć za kilkadziesiąt tysięcy (starych złotych) nazywałem ten tryb pracy „oszczędnościowym” (chyba tylko dla własnej kieszeni). Jak się wkrótce przekonasz drogi Czytelniku pierwsze praktyczne kroki z wykorzystaniem twego pierwszego systemu mikroprocesorowego warto będzie poczynić korzystając właśnie z tego trybu.

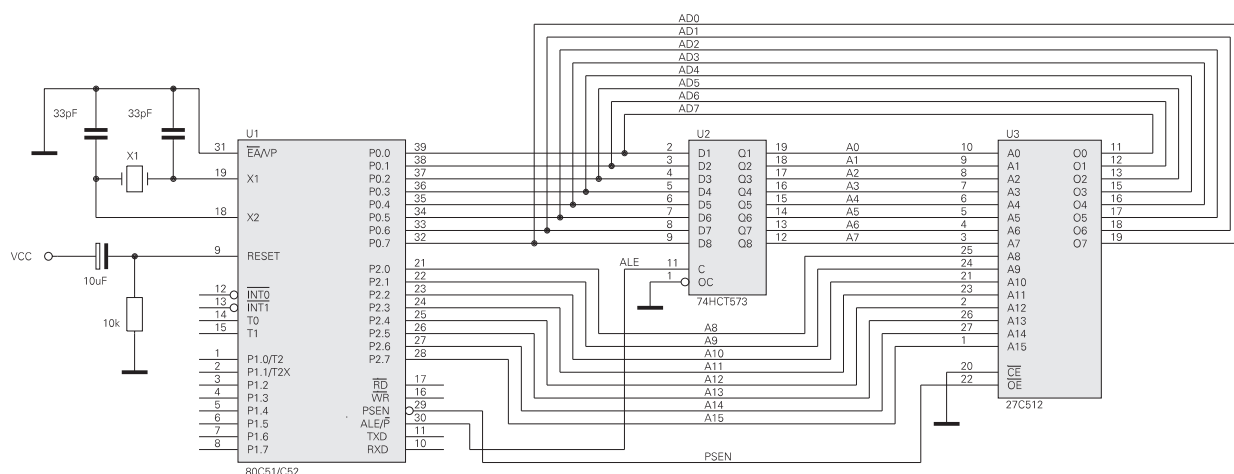
Tak jak przy nauce jazdy samochodem istnieje większe ryzyko stłuczki, tak

w przypadku mikroprocesora nie warto na początku ryzykować uszkodzeniem kostki z zapisanym w niej programem, lepiej jest kod programu umieścić w zewnętrznej pamięci EPROM.

A propos „pierwszego systemu z 8051” – w bieżącym numerze EdW znajdziesz opis kitu zawierającego wszystkie niezbędne elementy do zbudowania małego lecz w pełni funkcjonalnego systemu do nauki programowania i obsługi procesorów rodziny MCS-51. Publikacja tego systemu jest celowa i wyprzedza nieco rozpoczęcie nauki programowania. Wszystko po to abyś drogi Czytelniku miał trochę czasu na nabycie tego zestawu, zapoznanie się z nim, zmontowanie oraz wstępne sprawdzenie poprawnego działania. Wtedy wraz z pierwszym artykułem dotyczącym programowania 8051 będziesz mógł rozpocząć praktyczne lekcje z mikrokontrolerem. Początek kursu planowany jest w październiku.

Wracamy jednak do naszego artykułu. Pozostając wiernym zasadzie przekazywania Tobie, tylko niezbędnych, praktycznych informacji na temat '51-ki nie będę się rozwodził nad teorią obsługi zewnętrznej pamięci programu przez sam procesor, a jedynie przedstawię Ci niezawodny sposób dołączenia jej do układu mikrokontrolera.

Rysunek 5 przedstawia schemat elektryczny takiego połączenia. W układzie tym



Rys. 5. Sposób dołączenia zewnętrznej pamięci programu do procesora

zrezygnowano z użycia wewnętrznej pamięci programu – wykorzystano więc kostkę w wersji bez ROM (8051/52), pin /EA zwróto do masy, w jako zewnętrzną pamięć programu wykorzystano kostkę EPROM o pojemności 64kB typu 27C512. Oczywiście w praktyce nie jest konieczne używanie tak pojemnej pamięci. Do naszych celów i eksperymentów wystarczy pamięć 27C64 o pojemności 8kB. W takim przypadku linie adresowe A15, A14, A13 (piny 28, 27 i 26 portu P2) pozostaną niedołączone, lecz niestety i tak będą nie do wykorzystania, bowiem w trybie adresowania zewnętrznej pamięci programu (jak i danych) cały port P2 przekazuje starszy bajt 16-bitowego adresu.

Zgodnie z opisem z I części artykułu, w układzie wykorzystano dodatkowy rejestr w postaci układu 74HCT573, którego zadaniem jest zatrzaśnięcie młodszej części 16-bitowego adresu, czyli sygnałów A0...A7. W czasie trwania dodatkowego poziomu sygnału na wyjściu ALE, mikroprocesor wystawia: na piny portu P2 starszą część 16-bitowego adresu A8...A15, natomiast na port P0 wystawiona zostaje młodsza część adresu A0...A7, która zostaje od razu przekazana poprzez 74HCT573 na wejścia adresowe A0...A7 pamięci EPROM. Dzieje się tak dlatego że układ '573 aktywowany jest poziomem, toteż w przypadku gdy na jego wejściu C (połączonym z ALE) panuje stan wysoki, rejestr ten jest „przezroczysty” tzn. że dane pojawiające się na jego wejściach D1...D8 natychmiast pojawiają się na wyjściach Q1...Q8. Dołączenie wejścia /OC układu U2 do masy powoduje odblokowanie na stałe wyjść Q1...Q8

(gdy /OC=1 to wyjścia te przechodzą w stan wysokiej impedancji – lecz w naszym przypadku nie ma to praktycznego zastosowania).

Następnie podczas opadającego zbocza sygnału ALE młodsza część adresu zostaje

„zapamiętana” w U2, a procesor może wtedy z portu P0 odczytać kolejny bajt z pamięci EPROM podając stan niski na wyprowadzenie /PSEN, które jak widać ze schematu jest dołączone do wejścia /OE pamięci.

Wejście wyboru pamięci U3 /CE jest na stałe zwarte do masy, co oznacza że pamięć EPROM jest ciągle aktywna a do jej odczytu wystarczy niski poziom podany na wejście /OE.

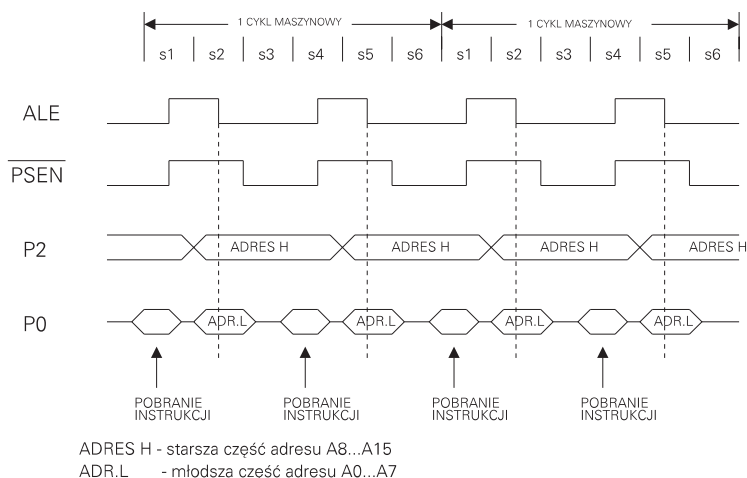
Czasem w różnych aplikacjach szczególnie w urządzeniach zasilanych bateryjnie wejście /OC układu U2 oraz wejście wyboru pamięci EPROM /CE są razem dołączone do wyjścia sygnału ALE. Nie jest to błędem, bowiem podczas cyklu odczytu z zewnętrznej pamięci tak programu jak i danych wszystkie operacje tak odczytu jak i zapisu do tych pamięci odbywają się podczas stanu niskiego na wyjściu ALE procesora.

Cykl odczytu z zewnętrznej pamięci programu możesz dodatkowo prześledzić na **rysunku 6**.

Podsumowując te akapit powinienś zapamiętać, że

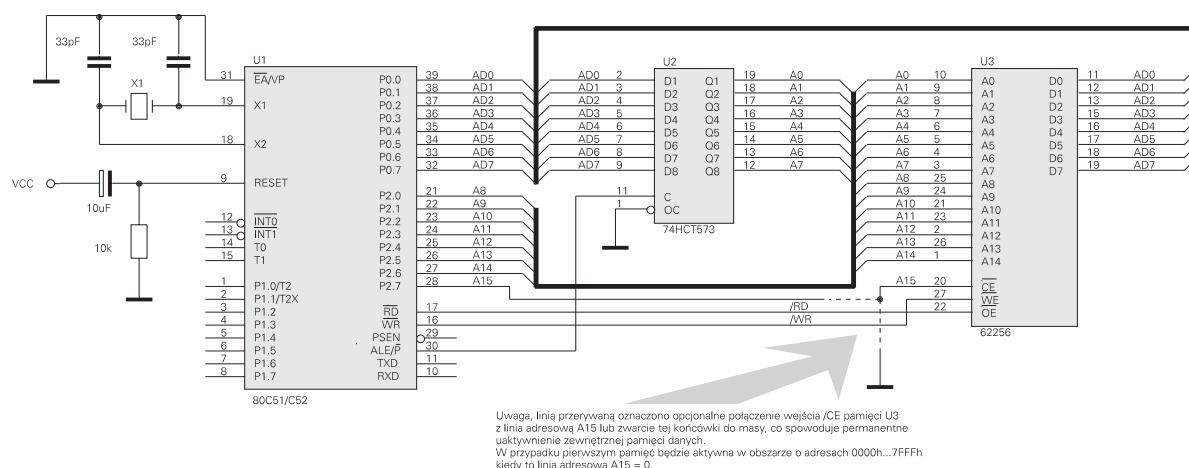
- przedstawiony sposób dołączenia zewnętrznej pamięci programu jest typowym dla tej konfiguracji pracy procesora
- odczyt kolejnych bajtów (rozkazów) z zewnętrznej EPROM odbywa się za pośrednictwem sygnału /PSEN, o którym pisałem w I części naszego cyklu przy okazji omawiania wyprowadzeń procesora
- dodatkowo w układzie z zewnętrzną pamięcią programu (jak się zaraz okaże także i danych) niezbędny jest dodatkowy układ scalony w postaci 8-krotnego rejestru równoległego 74HCT573. Można użyć wersji LS. Natomiast kostki 74HCT373 (LS373), które mają taką samą strukturę wewnętrzną, różnią się od układu 573 rozmieszczeniem wyprowadzeń D1...D8 i Q1...Q8.

Zainteresowanych w/w układami odsyłam do katalogów serii TTL. My w naszych rozwiązaniach będziemy nagminnie korzystać z układów '573 ze względu na bardziej korzystny rozkład jego wyprowadzeń wprost dostosowany do użycia



Rys. 6. Cykl odczytu z zewnętrznej pamięci programu

Też to potrafisz



Rys. 7. Sposób dołączenia zewnętrznej pamięci danych 32kB do procesora 8051

wraz z pocziwym 8051 oraz pochodnymi w obudowach DIL-40.

Zewnętrzna pamięć danych

W podobny sposób jak w przypadku zewnętrznej pamięci programu, do większości procesorów serii MCS-51 (w tym 8051/52) dołącza się zewnętrzną pamięć danych. Do tego celu używa się zazwyczaj (a prawie zawsze) kostek pamięci statycznych nazywanych często SRAM od skrótu: „Static RAM”, czyli pamięć statyczna. Pamięć taka powinna charakteryzować się 8-bitową organizacją danych oraz równoległym adresowaniem, tak jak w przypadku zwykłych EPROM-ów. Najpopularniejsze i spotykane w handlu, a przy tym najlepiej nadające się do naszych potrzeb kości pamięci SRAM to:

- a) 6116 – pamięć SRAM o pojemności 2kB (6116 – 16kbitów, czyli 16384 bitów / 8 = 2kB = 2048 B).
- b) 6264 – j/w lecz o pojemności 8kB (6264 – 64kbitów, czyli 65536 / 8 = 8kB = 8192 B).
- c) 62256 – j/w lecz o pojemności 32kB (62256 – 256 kbitów, czyli 262144 / 8 = 32kB...)

Cena układów wymienionych pamięci jest obecnie znikoma, nie znaczy to jednak że warto stosować pamięci z zapasem (np. największe 32kB), zawsze warto przewidzieć wielkość praktycznie wykorzystywanej ilości komórek pamięci i odpowiednio dobrać wymagany typ pamięci.

Pierwszy wymienionych układ SRAM 6116 spotykany jest w handlu w typowej obudowie 24-końcówkowej o rozstawie 600 mils, dwa ostatnie natomiast w obudowach 28-pinowych (600 mils, spotykane też są wersje w „wąskich” obudowach o rozstawie 300 mils).

Do kontrolerów serii '51 można dołączyć maksymalnie 64kB zewnętrznej pamięci danych. Można więc w takim (skrajnym) przypadku zastosować:

– 2 kostki 62256 (2x 32kB = 64kB) lub

– 8 kostek 6264 (8x 8kB = 64kB) lub w ostateczności

– 32 kostki 6116 (32x 2kB = 64kB).

Oczywiście dwie ostatnie sytuacje w przypadku wymaganej maksymalnej pojemności zewnętrznej SRAM są niepraktyczne, po pierwsze ze względu na ilość układów scalonych co prowadzi do zwiększenia płytki drukowanej, po drugie pojawia się konieczność stosowania dodatkowego dekodera adresu w postaci np. układu serii TTL-LS typ 74LS138 – dekodera 1 z 8.

Rysunek 7 pokazuje podstawowy układ dołączenia zewnętrznej kostki SRAM o pojemności 32kB do mikrokontrolera 8051. Jak widać układ połączeń jest bardzo zbliżony do schematu z rysunku 5, gdzie omawialiśmy praktyczny sposób dołączenia zewnętrznej pamięci programu typu EPROM. Dla zwiększenia czytelności rysunku rezygnujemy z rysowania każdego z połączeń magistrali adresowej (A0...A15) oraz danych (AD0...AD7) oddzielnie, w zamian zastąpimy je wspólnym „fachowym” symbolem „szyny” – w postaci pogrubionej kreski. Każde połączenie „odchodzące” od takiej magistrali ma swoją nazwę (etykieta), czyli że końcówki układów scalonych (w tym przypadku: procesora U1, pamięci U3 oraz zatrasku U2) przy których znajduje się taka sama etykieta literowa, są ze sobą elektrycznie połączone. Prawda, że prościej?!

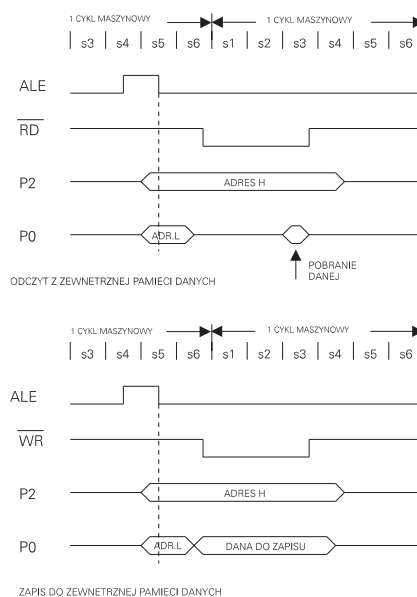
Przyzwyczajmy się zatem do takiego sposobu czytania i rysowania schematów, szczególnie jeżeli przedstawiony schemat zawiera wielokółkowe układy scalone takie jak mikroprocesor. Wracamy jednak do tematu.

Do adresowania pamięci wykorzystuje się, podobnie jak do pamięci programu, te same linie adresowe: A0...A15, oraz danych: AD0...AD7. Jednak sterowanie odczytem zewnętrznej pamięci danych zajmuje się sygnał

/RD (końcówka 17 – U1) od skrótu „Read” – przypomnij sobie część I naszego cyklu). Podanie odpowiednich stanów logicznych na końcówki adresowe pamięci (U3) odpowiadające adresowi komórki pamięci wewnętrznej układu, a następnie podanie stanu niskiego na końcówkę /RD spowoduje pojawienie się danej (z pamięci) na wyprowadzeniach magistrali AD0...AD7 i w konsekwencji odczyt jej przez mikroprocesor, za pośrednictwem portu P0.

Sytuację tę w postaci przebiegów przedstawia **rysunek 8**. Nie podaję tu czasów poszczególnych cykli, po pierwsze z czysto praktycznego powodu, po drugie ze względu że zależą one od częstotliwości zegara (rezonatora kwarcowego dołączonego do procesora).

Skoro wiemy jak praktycznie (sprzętowo) realizowany jest odczyt z zewnętrznej



Rys. 8. Cykl odczytu i zapisu z zewnętrznej pamięci danych

pamięci danych, to warto powiedzieć jak dokonywany jest zapis. Otóż do tego celu służy kolejny sygnał procesora nazywany /WR (od skrótu „write” - zapis), a wystawiany przez procesor na końcówkę 16 (U1) w czasie tej operacji. Podobnie ja w przypadku odczytu, najpierw procesor ustala adres na końcówkach A0...A15 (port P2 oraz wyjścia zatrasku U2), następnie podaje daną na port P0, po czym wystawia stan niski na pin /WR, co powoduje zapis danej w pamięci SRAM.

Jak już wcześniej wspomniałem, zastosowanie zewnętrznej pamięci tak programu jak i danych wiąże się ze zubożeniem typowej „jednokładowości” mikrokontrolera, bowiem zajęte zostają porty P0 i P2 procesora. O ile w przypadku pracy z zewnętrzną pamięcią programu dzieje się tak zawsze: port P0 pracuje jako multipleksowana (na zmianę) szyna danych lub młodszej części adresu, a port P2 wystawia starsze 8 bitów 16-bitowego adresu, o tyle w przypadku pracy z zewnętrzną pamięcią danych (bez zewnętrznej pamięci programu) istnieje możliwość oszczędniejszego gospodarowania portami procesora.

W przykładzie opisanym powyżej mówiliśmy, że procesor przy odczycie lub zapisie „....wystawia 16-bitowy adres (A0...A15) ...”. Do tego potrzebne są 2 porty, które w takim przypadku nie nadają się do dodatkowego wykorzystania, jako np. wyjścia sterowania przekazywanymi, lub czymkolwiek innym.

Istnieje jednak możliwość innego odczytu zewnętrznej pamięci danych, nazywana „stronicowaniem”. Ma ona zastosowanie szczególnie wtedy, kiedy zewn. pamięć danych ma mniejszy rozmiar od maksymalnej przestrzeni adresowej procesora np. 2kB. Jak widać z **rysunku 9** pamięć taka (U3) ma tylko 11 linii adresowych (A0...A10), co pozwala na zaadresowanie 2048 komórek pamięci (bajtów). Tak więc pozostałe linie adresowe procesora A11...A15 pozostałyby niewykorzystane gdyby zastosować

odczyt jak w poprzednim przypadku z pełnym adresem A0...A15.

Przy „stronicowaniu” „sposobie obsługi zewn. pamięci danych, procesor wystawia tylko młodszą część 16-bitowego adresu (linie A0...A7), zaś port P2 pozostaje „nietknięty”. Wnikliwy czytelnik zauważy że w konsekwencji takiego sposobu obsługi możliwe będzie zaadresowanie tylko 256 bajtów (2^8 do potęgi 8 = 256) tej pamięci, a nie jak w naszym przykładzie aż 2kB. No tak, chyba że przed odczytem przez procesor, sami, za pomocą sygnałów A8...A10 (wystawianych poprzez 3 piny portu P0) ustawimy niejako „fizyczny” adres 256 bajtowej strony adresowanej pamięci.

Zauważmy przecież że za pomocą tych trzech końcówek można „zaadresować” 8 stron po 256 bajtów każda co w sumie da nam do dyspozycji pełne 2048 bajtów, czyli 2kB. Zauważmy też że, co najważniejsze, pozostałe końcówki portu P2 pozostają wolne i możemy je dowolnie wykorzystać jako wejścia lub wyjścia cyfrowe... Prawda, że genialne?!

No tak, ale kiedy ten procesor adresuje pamięć za pomocą pełnego adresu, a kiedy za pomocą stronicowania! Otóż w celu rozróżnienia przedstawionych dwóch typów adresowania wprowadzone są dwie różne instrukcje procesora, których używa programista (w przyszłości Ty! drogi Czytelniku) podczas projektowania układu i pisania programu, w zależności od potrzeb, ale o tym powiemy dokładnie przy okazji nauki programowania 8051.

„Miksowanie” przestrzeni adresowych programu i danych

Często w praktyce podczas konstruowania i oprogramowywania układów wykorzystujących zewnętrzną pamięć tak programu jak i danych zdarzają się następujące dwie sytuacje:

- w zewnętrznej pamięci programu (EPROM – stałej) znajdują się, wprowa-

dzone przez programistę na etapie tworzenia programu, stałe np. w postaci tablic (sinusów, logarytmów niezbędnych do obliczeń) lub cokolwiek innego, nie będącego typową „instrukcją” programu lub jej argumentem. Aż się prosi żeby te dane umieszczane były w zewnętrznej pamięci danych (w wewnętrznej RAM prawdopodobnie zabrakłoby na nie miejsca), czyli w układach SRAM. Nie ma sprawy, można przecież je przenieść (programowo) z EPROM do SRAM w czasie inicjacji mikroprocesora, tylko po co !. Czy nie lepiej „jakoś” połączyć obszar pamięci programu i danych, zachowując oczywiście odrębne obszary adresowe, pozostawiając tylko jednolity sposób odczytu tych danych?!

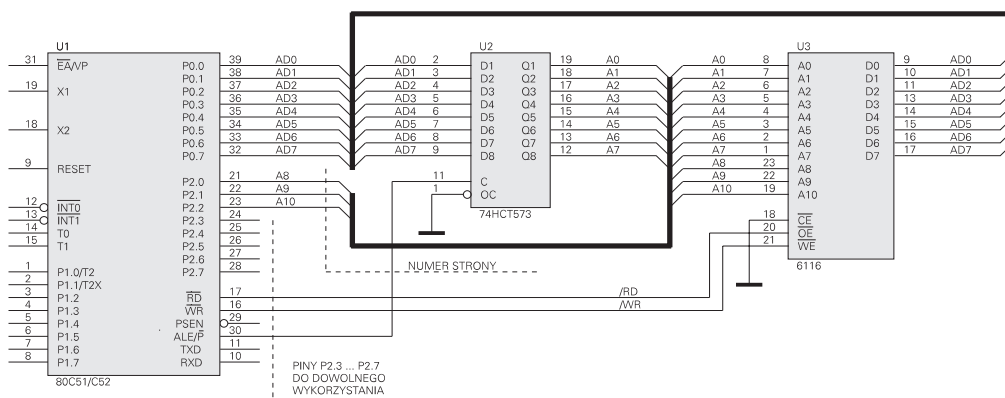
I druga sytuacja, bardziej praktyczna w fazie nauki sytuacji.

- Zbudowałem uniwersalny mikrosterownik z 8051..., chciałbym mieć możliwość wpisywania na bieżąco (np. za pomocą komputera) moich programów i testowania ich w tym układzie bez potrzeby każdorazowego programowania pamięci EPROM za pomocą drogiego programatora, na który mnie zresztą nie stać Czy nie dało by się wykorzystać dołączoną pamięć danych SRAM jako część obszaru adresowego pamięci programu (i odwrotnie) i wpisywać do niej nowe programy bez wyjmowania układu scalonego z podstawki ?...

W przedstawionych dwóch przypadkach, rozwiązaniem problemu jest wykorzystanie do odczytu danych tak z pamięci programu jak i danych, iloczynu sygnałów:

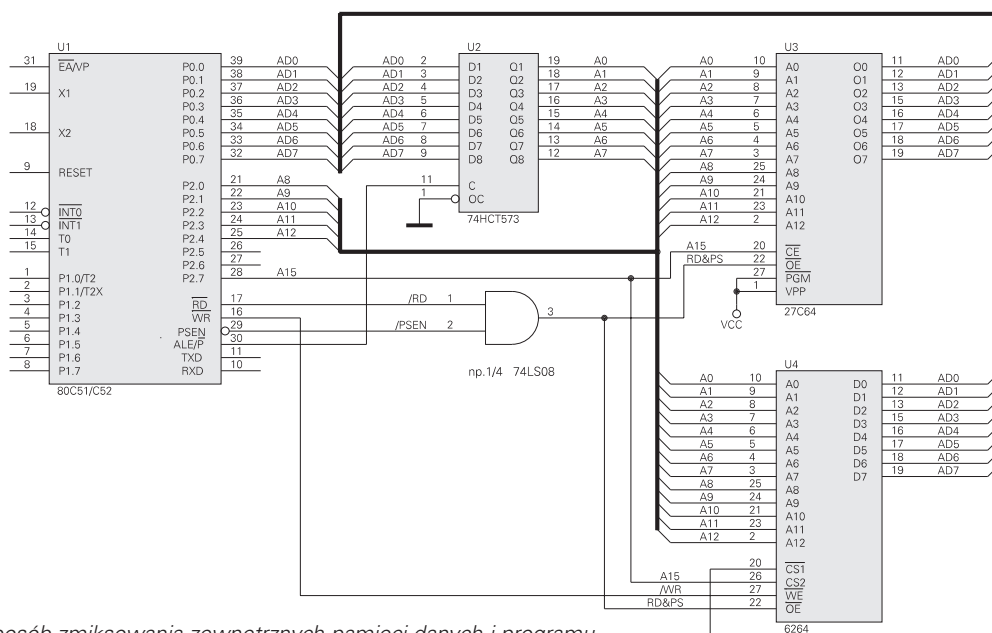
- odczytu z zewnętrznej pamięci danych /RD oraz
- odczytu z zewnętrznej pamięci programu /PSEN

Iloczyn taki w praktyce najłatwiej jest wykonać za pomocą pojedynczej 2-wejściowej bramki AND, jak pokazano na **rysunku 10**. Przedstawiono nieco rozbudowaną konfigurację procesora pracującego



Rys. 9. Przykład adresowania zewnętrznej pamięci danych 2kB poprzez stronicowanie

Też to potrafisz



Rys. 10. Sposób zmkisowania zewnętrznych pamięci danych i programu

z zewnętrzną pamięcią programu U3 (EPROM 8kB) oraz pamięci danych U4 (SRAM 8kB). Zauważmy jednak że odczyt z obu pamięci może odbywać się „tak samo”. Dzięki wymnożeniu sygnałów /RD i /PSEN, procesor ma możliwość odczytu kolejnej instrukcji programu (za pomocą /PSEN) tak z pamięci U3 jak i U4, bowiem sygnał RD&PS (będący iloczynem /RD i /PSEN) steruje wejściami /OE odczytu obu pamięci (U3 i U4). Oczywiście sprawa zapisu nadal dotyczy tylko pamięci SRAM U4 (dołączony sygnał /WR procesora U1 do końcówki /WE pamięci U4).

W celu rozdzielienia obszarów adresowych obu pamięci (tak aby nie następował konflikt przy odczycie na magistrali AD0...AD7, co może mieć miejsce w przypadku jednoczesnego pojawienia się danej na wyjściach 11...19 układu U3 i U4), wykorzystano linię adresową A15. Dzięki niej pamięć EPROM (U3) będzie aktywna kiedy sygnał A15 będzie miał stan niski, co jest adekwatne do obszaru adresowania równemu: 0000h...7FFFh (oczywiście wtedy pamięć U4 jest nieaktywna, bo sygnał wyboru CS2 – U4 ma stan linii A15 = niski).

Pamięć SRAM U4 będzie natomiast obsługiwana w obszarze adresowym dla A15 równego logicznie „1”, czyli w zakresie: 8000h...FFFFh.

Na zakończenie, krótki przykład w jaki sposób w takim układzie (z rys.10) realizowany jest przypadek b) omawiany wcześniej.

Otóż w pamięci EPROM U3 na stałe znajduje się program umożliwiający np. przesłanie danych (którymi mogą być także instrukcje programu) z komputera do

naszego układu mikroprocesorowego, chociażby tego z rys.10. Program ten często zwany „monitorem” lub „biosem” wykonywany zostaje zaraz po włączeniu zasilania procesora, bowiem pamięć EPROM z rys.10 – układ U3 zaczyna się od adresu początkowego 0000h.

Nie jest na razie istotne w jaki sposób wspomniane dane zostają przesłane z komputera, (może być w tym celu użyty port szeregowy lub np. równoległy komputera).

Ważne jest że po przesłaniu mogą one być zapisane w zewnętrznej pamięci SRAM – U4, która przecież daje się zapisywać jak zwykłą pamięć danych dzięki sygnałowi /WR.

Kiedy już te dane, będące np. kawałkiem jakiegoś nowego, utworzonego przez Ciebie programu (powiedzmy zegarka z 256 alarmami i tyłoma timerami) zostaną wpisane do pamięci U4, możesz teraz rozkazać procesorowi (wydając odpowiednią instrukcję), „skoczyć” do adresu 8000h i stamtąd rozpocząć dalsze wykonywanie programu. Co będzie w efekcie ...?, w efekcie dzięki bramce AND (rys.10) jak powiedziałem wcześniej, dalej wykonywanym programem będzie ten przesłany wcześniej z komputera i umieszczony w SRAM, czyli np. twój super-zegarek (lub cokolwiek co w przyszłości stworzysz).

Zaletami takiego rozwiązania są:

- brak konieczności stosowania drogich programatorów EPROM
- bardzo szybkie ładowanie nowych programów do pamięci SRAM, co pozwoli uczyć się programiście (takemu jak Ty) na szybkie obserwowanie efektów swojej pracy

- niepotrzebne jest oczywiście kasowanie SRAM przed zapisem nowej wersji tworzonego programu

Nie brakuje i wad:

- po wyłączeniu zasilania program umieszczony w SRAM zostaje wymazany (ale dalej znajduje się np. w komputerze);
- konieczne jest „posiadanie” EPROM ze wspomnianym „biosem” (monitorem).

Jak się wkrótce przekonasz, drogi Czytelniku, na etapie nauki programowania i obsługi procesorów 8051 pierwsza „wada” w praktyce wcale nie jest wadą. O drugi drobiazg (zaprogramowany EPROM) nie musisz się martwić, będziesz mógł go otrzymać wraz zestawem dla początkujących programistów, który znajdziesz w tym numerze EdW.

Bardziej ambitnych muszę w tym miejscu uspokoić. Jestem pewien, że każdy kto ukończy nasz kurs projektowania będziecie w stanie sami napisać swój własny program monitora. Na razie na etapie wstępnej nauki programowania, do której niebawem przejdziemy, niezbędna będzie wersja „gotowa” biosu, wierzcie mi drodzy Czytelnicy, zaoszczędzi to wam wielu stron teorii, która na początku mogłaby niejednego z Was przerazić swoimi rozmiarami, a którą sami nabędziecie z czasem, ucząc się pisać pierwsze swoje programy.

W następnym odcinku omówię pokrótce (praktyczne rady) ważniejsze tematy dotyczące „czasowego” sposobu wykonywania przez mikrokontroler 8051 programu, po czym przejdziemy do pierwszych kroków w programowaniu.

Sławomir Surowiński



W kolejnym odcinku opisującym procesor 8051 przedstawię kilka praktycznych wiadomości na temat pracy mikrokontrolera, jej szybkości w zależności od częstotliwości taktującej zegara. Następnie pokrótce zapoznamy się z niemniej ważnym blokiem funkcjonalnym, czyli: układem czasowo – licznikowym. Tak jak w poprzednich odcinkach nie będę wgłębiał się w teorię dotyczącą tych zagadnień, przekazując jedynie wiadomości potrzebne do osłuchania a raczej oswojenia się z procesorem. Szczegółowy opis wszystkich rejestrów sterujących pracą 8051 opiszę w jednym z kolejnych odcinków, który będzie niejako podstawową „ściągawką” w stawianiu pierwszych kroków w programowaniu.

Zegar systemowy

O zegarze wspominałem na początku naszego cyklu, kiedy to poznawaliśmy wyprowadzenia mikrokontrolera. Czy pamiętasz funkcje jaka pełnią wyprowadzenia 18 i 19 procesora (w obudowie DIL-40)? Jeśli nie to radzę przeczytać stosowny akapit w EdW 5/97 str. 41.

Powiedziałem, że do poprawnej pracy, a w zasadzie w ogóle do pracy, czyli do „poruszenia” całego procesora potrzebny jest zewnętrzny obwód oscylatora. W praktyce taki obwód realizuje się dołączając zewnętrzny rezonator kwarcowy o częstotliwości z zakresu od mniej więcej 1MHz do 16MHz. W handlu spotyka się także wersje 8051 mogące pracować w szerszym zakresie: od pojedynczych „Hz” (wersje całkowicie statyczne CMOS) do nawet: 33...40MHz. W każdym przypadku producent konkretnego modelu procesora umieszcza na jego obudowie oprócz nazwy układu także symbol liczbowy określający maksymalną częstotliwość zewnętrznego sygnału taktującego (która w praktyce jest także równa dołączanemu rezonatorowi kwarcowemu). Nie będziemy jednak się zajmować tutaj sposobami oznaczania poszczególnych wersji 8051, nie to jest w tej chwili najważniejsze, ważne jest to że im większa częstotliwość graniczna procesora, tym oczywiście układ będzie mógł pracować szybciej.

Nie oznacza to, że w każdym przypadku „uganiać się” będziemy za możliwie najszybszą wersją układu, nie zawsze przecież taka będzie potrzebna. Wiedzieć wszakże trzeba że tak jak w przypadku wszystkich układów wykonanych w technologii MOS (CMOS, HMOS) wraz ze wzrostem częstotliwości pracy układu wzrasta wydzielana w nim moc, czyli wzrasta pobierany przez procesor prąd ze źródła zasilania.

Praktyczny wniosek nasuwa się sam – jeżeli masz zamiar zastosować procesor w układzie przenośnym zasilanym np. z niewielkiej baterii, z pewnością nie użyjesz procesora 8051 w wersji 20MHz! Poborem mocy oraz zagadnieniami z tym związanymi zajmujemy się w dalszej części naszego cyklu.

Przejdźmy jednak do informacji praktycznych.

Musisz wiedzieć, że częstotliwość (nazwijmy ją jako F_{xtal}) uzyskiwana z rezonatora kwarcowego (dołączonego do pinów 18 i 19) jest we wnętrzu procesora kilkakrotnie dzielona. I tak w praktyce spotkasz się w literaturze i katalogach na temat 8051 i podobnych z pojęciami jak:

- a) dwufazowy sygnał taktujący procesor (F_s) – sygnał powstały z podzielenia przez 2 częstotliwości oscylatora (np. przy kwarcu = 12 MHz, F_s = 6MHz). Sygnał ten używany jest bezpośrednio do taktowania układów wewnętrznych procesora i nie jest dostępny na żadnym z zewnętrznych jego wyprowadzeń.

- b) sześć cykli sygnału F_s składa się na tzw. cykl maszynowy procesora, czyli okres wykonywania elementarnej czynności (jakiej?... za chwilę) przez naszą kostkę. Z prostych obliczeń wyniknie Ci że, cykl maszynowy zajmuje: $F_s \times 6 = 2 \times F_{xtal} \times 6 = 12$ cykli oscylatora, czyli dla np. $F_{xtal}=12\text{MHz}$ będzie to 1MHz.

Cykl maszynowy jest bardzo ważnym pojęciem, z jego częstotliwością ($F_{xtal}/12$) zachodzą podstawowe czynności procesora takie jak:

- pobieranie kodu rozkazów (czy to z wewnętrznej pamięci programu, czy z zewnętrznej)
- wykonywanie instrukcji programu
- pobieranie danych z zewnętrznej pamięci (jak i z wewnętrznej)
- zwiększanie wartości wbudowanych liczników: T0, T1 także T2 dla 8052
- próbkowanie wejść zewnętrznych przerwań: INT0 i INT1
- wystawianie sygnału ALE – pin 30 (opisywanego wcześniej) niezbędnego do zapisu młodszej części adresu multiplexowanej szyny adresowej – portu P0, przypomnij sobie wcześniejszy odcinek.

Z częstotliwością tą taktowany jest także wbudowany port szeregowy w specjalnym trybie ustawionym przez użytkownika programowo.

Warto wiedzieć że cykl maszynowy dzieli się także na fazy (po 6 na każdy cykl), jednak ich opis i znaczenie w praktyce przy

Też to potrafisz

konstruowaniu większości urządzeń jest niepotrzebne, dlatego nie będziemy się tym w bieżącym odcinku zajmować.

Rysunek 1 ilustruje zależności czasowe pomiędzy omówionymi podstawowymi pojęciami spotykanymi podczas omawiania zegara i cyklu maszynowego. Warto o tym pamiętać.

Układy czasowo-licznikowe

Pod pojęciem tym kryją się wielokrotnie wspomniane dwa 16-bitowe liczniki T0 i T1 oraz dodatkowy licznik T2 który występuje w procesorze 8052.

Najogólniej mówiąc każdy z tych liczników a właściwie układów czasowo-licznikowych jest tak uniwersalnym blokiem że z wykorzystaniem jego można dokonać następujące dwie podstawowe operacje:

a) za pomocą T0 (T1 lub T2) można zliczać impulsy z zewnętrznego wejścia licznikowego; pin 14 dla T0 i pin 15 dla T1 (tryb licznika) oraz

b) można zliczać wewnętrzne impulsy pochodzące z układu taktującego procesor, w każdym przypadku będzie to sygnał o częstotliwości $= F_{xtal} / 12$. Czyli jeżeli „dopieliliśmy” do mikrokontrolera kwarc o częstotliwości 6 MHz to częstotliwość sygnału taktującego licznik T0 lub T1 (T2) będzie równa $6 \text{ MHz} / 12 = 500 \text{ kHz}$. W tym trybie zwanym czasomierzem, liczniki wykorzystuje się do odmierzania pewnych określonych programowo przez użytkownika odcinków czasu (opóźnień) i generowania przerwań po przepelnieniu któregoś z liczników.

W dalszej części artykułu przedstawię przykład takiego zastosowania.

W przypadku wykorzystania układu licznikowego w obu przypadkach należy wiedzieć że:

– maksymalna liczba zliczonych impulsów jest określona pojemnością 16-bitowego licznika, czyli 2 do potęgi $16 = 65536$ (licznik zlicza od 0 do 65535 po czym po nadejściu kolejnego impulsu jest zero-

wany oraz z zależności od potrzeb jest generowane odpowiednie przerwanie);

– licznik można w dowolnym momencie uruchomić (zezwolić na zliczanie) lub zatrzymać wydając w programie odpowiednią komendę;

– do licznika można w każdej chwili wpisać dowolną wartość (16-bitowa liczba), co spowoduje że licznik będzie zliczał impulsy od tej wartości aż do przepelnienia; wpisu takiego najlepiej jest dokonywać w czasie gdy licznik jest zatrzymany;

– dodatkowo licznika można „bramkować” czyli uzależnić jego pracę lub zatrzymanie w zależności od stanu panującego na wejściach: INT0 dla licznika T0 oraz INT1 dla licznika T1;

– oprócz tego licznik T1 (jak i T2 w 8052) może „taktować” wbudowany port szeregowy w specjalnym trybie który opiszemy przy okazji omawiania tego bloku procesora.

W przypadku używania liczników do zliczania impulsów zewnętrznych należy wiedzieć, że maksymalna częstotliwość (F_{max}) zliczanych impulsów jest ściśle zależna od częstotliwości oscylatora kwarcowego F_{xtal} i określona jest zależnością:

$$F_{max} = \frac{F_{xtal}}{24}$$

Czyli w przypadku zastosowania kwarcu o częstotliwości 12 MHz maksymalna częstotliwość impulsów na wejściu licznika może wynieść 500kHz dodatkowo przy założeniu że przebieg ma wypełnienie 1:2. Ograniczeni wynika z faktu, że liczniki T0 i T1 (także T2) fizycznie nie wyglądają jak np. 7493, lecz zliczają na zasadzie „próbkowania” wejścia impulsów w celu stwierdzenia czy jest logiczne „0” a następnie „1”. Operacja ta odbywa się na synchronicznie z cyklem maszynowym o którym mówiliśmy wcześniej. W każdym cyklu maszynowym procesor próbuje wspomniane wejścia liczników, to też stwierdzenie, że na jednym z wejść sygnał zmienił wartość z „0” na „1” lub odwrotnie zajmuje 2 cykle maszynowe – stąd bierze się F_{max} .

Przy projektowaniu układów dla bezpieczeństwa warto jednak granicę tę nieco obniżyć (np. do 480 kHz przy $F_{xtal}=12\text{MHz}$).

Podane możliwości wykorzystania liczników jako np. wejść mierzących częstotliwość nie są zbyt imponujące. Jednak jak się sam przekonasz w przyszłości takie wykorzystanie liczników procesora np. do bezpośredniego mierzenia częstotliwości wejściowej jest niepraktyczne. Dlatego, w odpowiedzi podaję przykład.

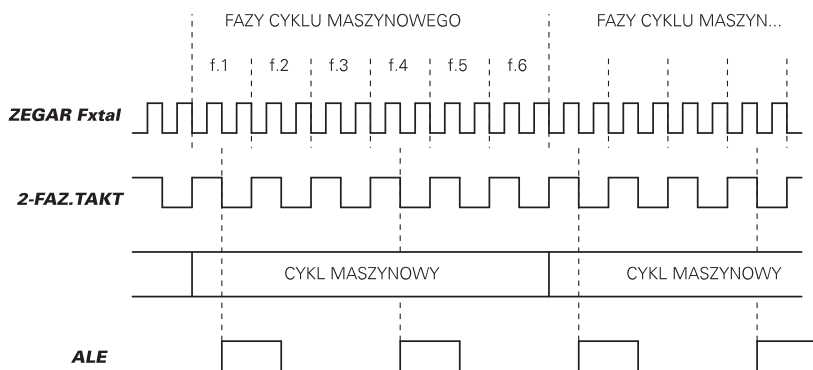
Wyobraź sobie że chcesz zmierzyć częstotliwość rzędu kilku (kilkunastu MHz) a więc znacznie przekraczającą możliwości liczników procesora. Do mierzenia każdej częstotliwości w zwykłych miernikach wykorzystuje się dwa sygnały: mierzony i oczywiście bramkujący. Ten drugi pochodzi zazwyczaj z wbudowanego w przyrząd generatora wzorcowego i powstaje przez wielokrotne podzielenie częstotliwości generowanej najczęściej za pomocą rezonatora lub generatora kwarcowego.

Nasuwa Ci się pewnie w tej chwili myśl: „...” No dobrze wykorzystam generator procesora a właściwie jeden z jego liczników do odmierzania czasu bramkowania a drugim licznikiem mierzę impulsy wejściowe i będę miał szukaną częstotliwość, tylko że kilka „MHz” to stanowczo zbyt wiele na mój procesor „...”

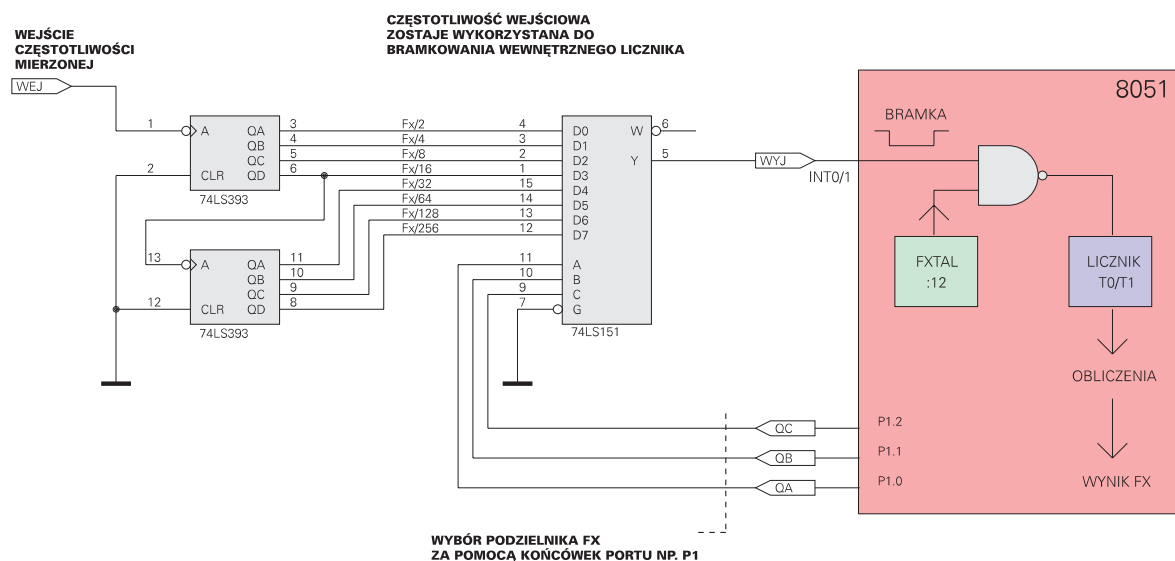
A gdyby tak odwrócić role i zliczać impulsy wewnętrzne o częstotliwości przeciwznej i równej $F_{xtal} / 12$, a sygnał wejściowy wielokrotnie podzielić i wykorzystać do bramkowania licznika. Wtedy w zasadzie otrzymamy nie częstotliwość ale okres przebiegu wejściowego, ale od czego mamy mikroprocesor, który potrafi wykonywać obliczenia arytmetyczne – logiczne. Potrafi on także dokonać odwrócenia wyniku okresu w efekcie czego otrzymamy liczbowa wartość mierzonej częstotliwości. Tak więc w prosty sposób można dokonać pomiaru dowolnej częstotliwości wejściowej a przy okazji wyświetlić ciekawskiemu użytkownikowi także okres badanego przebiegu.

Rysunek 2 ilustruje zasadę pomiaru częstotliwości, którą najczęściej wykorzystuje się w układach z mikroprocesorem. Wbrew pozorom metoda ta daje świetne wyniki oraz pozwala uzyskać dużą dokładność pomiaru przy krótkich czasach bramkowania.

Należy tylko częstotliwość wejściową podzielić przez taką wartość która da wynik zbliżony do wymaganego okresu bramkowania. Dodatkowy sprzętowy programowany dzielnik najprościej jest wykonać chociażby za pomocą kaskadowo połączonych 4-bitowych liczników binarnych 7493 lub podwójny 74393 wraz z multiplexerem np. 74151 (sprawdzić w katalogu). Wejścia multiplexera decydujące o stopniu podziału sterowane będą oczywiście z wolnych końcówek dowolnego portu mikroprocesora 8051 (np. z P1).



Rys. 1. Zegar, faza, a cykl maszynowy procesora 8051



Rys. 2. Zasada pomiaru częstotliwości z wykorzystaniem mikroprocesora

Uzyskany na wyjściu multiplexera przebieg doprowadzony zostanie do wejścia bramkującego licznik INT0 – dla licznika T0 lub INT1 kiedy zlicza licznik T1. W tym przykładzie oczywiście licznik będzie zliczał impulsy wewnętrzne, tak więc poszukiwana częstotliwość można będzie obliczyć z proporcji

$$\frac{F_x}{F_{wew}} = \frac{L_x}{L_{wew}}$$

gdzie:

F_x – częstotliwość szukana

F_{wew} – częstotliwość imp. wewnętrznych = $F_{xtal} / 12$

L_x – liczba zliczonych impulsów z zewnątrz

L_{wew} – liczba impulsów zliczonych przez licznik wewnętrzny

W przypadku kiedy sygnał mierzony wykorzystujemy jak w naszym przykładzie do bramkowania to $L_x = 1$ toteż po przekształceniu otrzymamy:

$$F_x = \frac{1}{L_{wew} \cdot F_{wew}}$$

ale $L_x = 1$ to

$$F_x = \frac{F_{wew}}{L_{wew}}$$

Wynik należy jeszcze pomnożyć przez wartość dzielnika sprzętowego, czyli wzór przyjmie postać:

$$F_x = D_z \cdot \frac{F_{wew}}{L_{wew}} \text{ [Hz]}$$

gdzie D_z to dzielnik pamiętając że

$$F_{wew} = \frac{F_{xtal}}{12}$$

gdzie F_{xtal} – częstotliwość oscylatora.

Dla przykładu założmy że zewnętrzny programowany dzielnik dzieli częstotliwość mierzoną przez 64 tak, że na wyjściu multiplexera otrzymujemy przebieg którego okres, a więc czas od jednego np. ujemnego zbocza do drugiego ujemnego zbocza, wynosi tyle, że wewnętrzny licznik procesora bramkowany tymi zboczami zliczy w czasie jednego okresu 54532 impulsy (których częstotliwość przy zastosowaniu kwarcu 12MHz wynosi 1MHz). Skomplikowane? Przeczytaj to zdanie jeszcze raz i postaraj się zrozumieć. No i jak z tego policzyć częstotliwość? Z naszego wzoru! Tak więc:

$$F_x = \frac{64(\text{dzielnik}) \cdot 1000000}{54532} = 1173 \text{ Hz}$$

Sprawdźmy na „chłopski rozum” czy aby wynik jest w porządku:

- licznik procesora w przeciągu jednego okresu podzielonego przebiegu wejściowego zliczył 54532 impulsy każdy po 1us (mikrosekundzie) przy zegarze 12MHz
- tak więc okres podzielonego przebiegu z wejścia wynosi: 54532 us (mikrosekundy), czyli 54,532 ms (milisekundy).
- odwracamy tę wartość i uzyskujemy liczbę: 18,3378
- pamiętając o dzielniku wstępnym mnożymy otrzymaną liczbę przez niego czyli przez 64
- otrzymujemy wynik:

$$F_x = 18.3378 \cdot 64 = 1173 \text{ Hz}$$

tak więc się zgadza.

Uff, jeżeli masz dość obliczeń, odpocznij chwilę, w każdym razie musisz pamiętać że z wykorzystaniem procesora możliwości obliczeniowe oraz atrakcyjność przyrządu pomiarowego wzrasta, pomi-

mo pozornie trudniejszego sposobu mierzenia tej wielkości.

Po tym małym przykładzie wracamy do tematu naszego odcinka.

Fizycznie 16-bitowe liczniki T0, T1 i T2 są zbudowane z dwóch 8-bitowych „połówek”, do których programista ma dostęp na poziomie programu. W czasie zliczania impulsów przeniesie z młodszego bajtu licznika nazywanego jako TL powoduje automatyczną inkrementację bajtu starszego TH, przy jednoczesnym wyzerowaniu bajtu TL. Taka sytuacja przedstawia jeden z kilku trybów w którym dwie połówki stanowią całość – 16-bitowy licznik. W mnemonice (nazewnictwie) 8051 wspomniane dwie połówki liczników mają swoje oznaczenia, i tak: dla licznika T0 są to TH0 i TL0 (starsza i młodsza część), dla licznika T1 – TH1 i TL1. Podobnie jest w przypadku licznika T2 w procesorze T2, gdzie mamy: TH2 i TL2.

W praktyce użytkownik ma możliwość zaprogramowania liczników w kilku innych trybach pracy, nie mniej użytecznych. W sumie jest ich 4, nazywane potocznie: trybem 0, 1, 2 i 3. Poniżej opiszemy je po krótko. Ze względu na to że liczniki T0 i T1 są bliźniaczo podobne będziemy odwoływać się przy opisie tylko do licznika T0. Ze względu na rozbudowane funkcje licznik T2 zostanie opisany w dalszej części artykułu osobno.

Tryb 0

W tym trybie licznik pracuje w konfiguracji 13-bitowej. Starszy bajt TH0 zawiera 8 bardziej znaczących bitów licznika (bity 7...0 TH0), natomiast 5 pozostałych bitów to najstarsze bity z TL0 (bity 7...3). Trzy najmłodsze bity bajtu TL0 są nieistotne i ignorowane przez procesor. **Rysunek 3**

Też to potrafisz

przedstawia konfigurację licznika T0 (T1) w tym trybie.

Do licznika (do bajtów TH0 i TL0) można wpisać dowolną wartość pamiętając, że 3 najmłodsze bity słowa TL0 będą ignorowane. Licznik po uruchomieniu będzie zliczał od wartości wpisanej na początku (może to także być wartość „0”) do wartości maksymalnej czyli $2^{13-1}=8191$ po czym się wyzeruje, dodatkowo zgłaszając jeżeli potrzeba przerwanie informując program o tym fakcie. Jeżeli niektórych z Was denerwuje te „przerwanie” to nie będę trzymał w niepewności i wyjaśnię na czym polega istota zgłoszenia przerwania w momencie przepełnienia licznika.

Wyobraź sobie że chcesz odmierzać równomierne odstępy czasu o długości np. 2,45 ms (milisekund). Co robisz? Mając do dyspozycji procesor z kwarcem np. 12 MHz wpisujesz do licznika T0 wartość początkową równą:

wartość_maksymalna_licznika_T0 + 1 = 2450

Dlaczego 2450?

Bo jest to $2450 \times 1 \mu s = 2,45 \text{ ms}$

A skąd $1 \mu s$?

Bo przecież jest to częstotliwość zliczania wewnętrznych impulsów = $F_{xtal} / 12$.

Tak więc po odjęciu otrzymasz liczbę: $8191 + 1 = 2450 = 5742$, którą zanim wpiszesz do licznika T0 musisz pomnożyć dodatkowo przez 8 (1000 binarnie) bo pamiętaj przecież że trzy najmłodsze bity 2..0 TL0 są ignorowane.

Teraz po wpisaniu uruchamiasz licznik, który liczy w górę od wartości wpisanej: 5742 aż do przepełnienia – 8191 czyli 2450 impulsów, zajmie mu to więc dokładnie 2450 us – czyli 2,45 ms, po czym zgłoszone zostanie przerwanie, w którym Ty – przyszły programista określisz co akurat ma się wydarzyć po upływie 2,45 ms.

Tryb 1

Tryb ten jest bardzo podobny do trybu 0, z tym że do zliczania wykorzystywane są wszystkie 16-bitów licznika. Stąd nasuwa się wniosek że maksymalną pojemność licznika w tym trybie wynosi 65535, po czym następuje przepełnienie czyli wyzerowanie z ustawieniem znacznika zgłoszenia przerwania (jeżeli jest taka potrzeba).

Tryb ten najczęściej wykorzystuje się do generowania przerwań mających na celu odmierzenie czasu np. przy zegarze czasu rzeczywistego. Przykład zastosowania może być taki jak poprzednio zwiększa się tylko zakres mierzonych odstępów czasu.

Tryb 2

Nieco ciekawszy jest tryb 2, w którym pracuje tylko młodsza połówka 16-bitowego licznika a więc TL0 (TL1 dla licznika T1). Ośmiobitowy licznik TL0 zlicza w górę aż do wartości maksymalnej czyli 255, po czym ... tu uwaga! Automatycznie zostaje przepisana do niego wartość początkowa ze starszej połówki TH0. Tak więc raz wpisując do TH0 jakąś wartość, nie musimy się martwić aby zrobić to programowo powtórnie przy przepełnieniu pracującego licznika – TL0.

Tryb ten ma wiele zastosowań, szczególnie przydaje się tam gdzie potrzebne jest generowanie przerwań w równych odstępach czasu, np. przy generacji sygnału prostokątnego o zadanej częstotliwości i wypełnieniu. O tym jak to się realizuje dowiesz się Czytelniku podczas nauki programowania 8051. Na razie proponuję Ci się nad tym zastanowić samemu. Pomocny będzie **rysunek 4** na którym pokazana jest struktura licznika w trybie 2.

Warto także wiedzieć, że tryb ten w liczniku T1 wykorzystuje się do taktowania portu szeregowego procesora,

a właściwie do określenia szybkości transmisji danych przez ten port. Wtedy jednak licznik nie może spełniać innych funkcji, np. generować przerwań przy przepełnieniu.

Tryb 3

Tryb ten dotyczy obu liczników T0 i T1 procesora na raz. Otóż w trybie tym licznik T1 jest zatrzymany i nie pracuje. Dwa bajty licznika T0: TH0 i TL0 pracują jako dwa niezależne 8-bitowe liczniki, przy czym istnieją pewne ograniczenia co do ich funkcji, a mianowicie:

- TL0 może liczyć impulsy z wejścia T0 lub pracować jako czasomierz zliczając impulsy wewnętrzne ($F_{xtal} / 12$)

- TH0 może pracować tylko jako czasomierz, czyli zliczać impulsy wewnętrzne

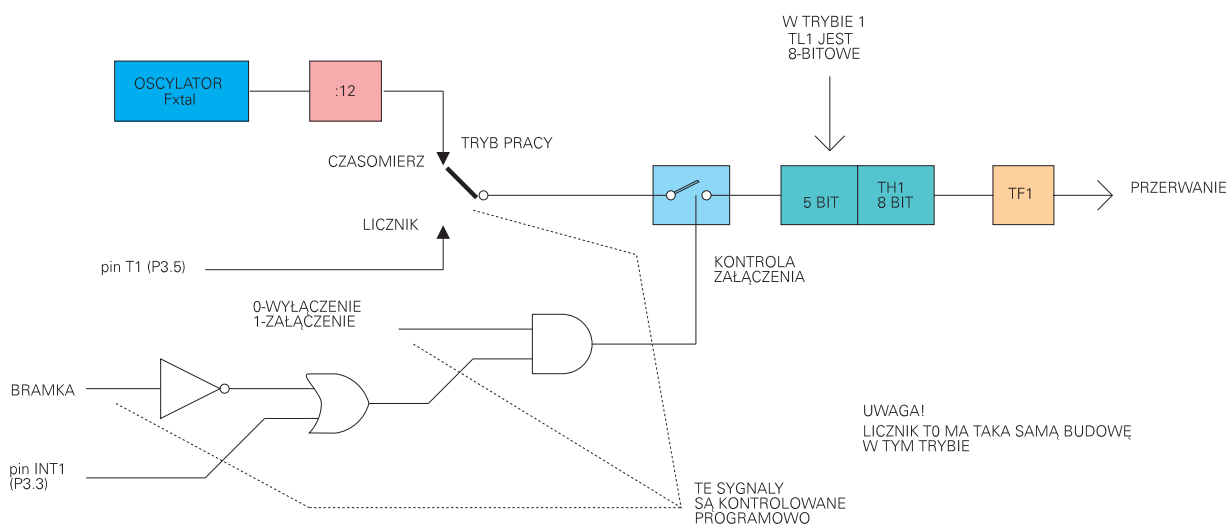
Tryb ten został zaimplementowany przez twórców 8051 po to, aby w wypadkach kiedy licznik T1 używany jest do określania szybkości transmisji poprzez port szeregowy, a programiście niezbędne są dwa dodatkowe liczniki, których role spełniają wtedy wspomniane TL0 i TH0.

W obecnych czasach, jeżeli zachodzi taka potrzeba, czasem lepiej jest zastosować procesor w wersji 80C52 z wbudowanym trzecim licznikiem T2. Niemniej jednak warto wiedzieć o tym nietypowym trybie liczników T0 i T1.

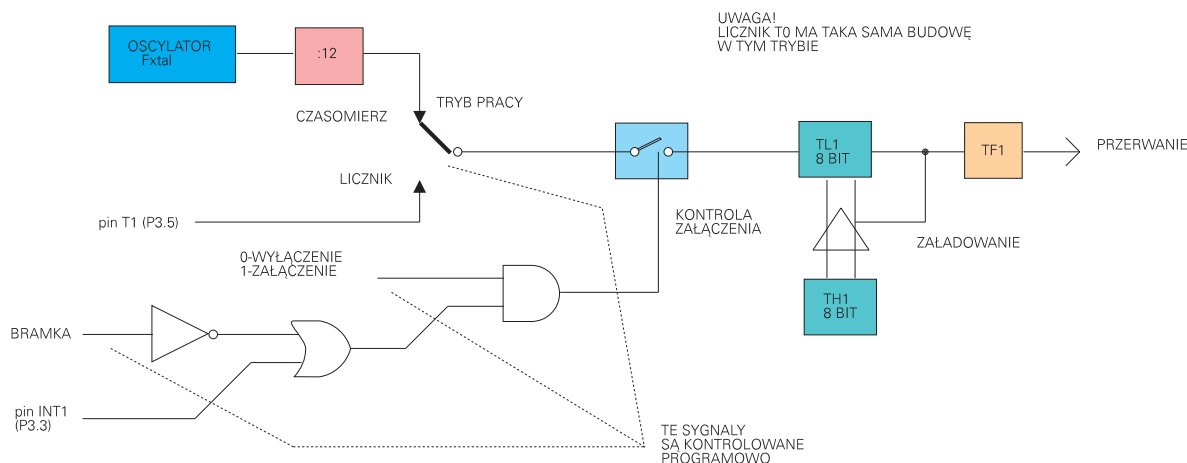
Licznik T2 w kostce 80C52

W mikrokontrolerze 80C52 występuje dodatkowy licznik nazywany T2. Podobnie jak licznik T0 i T1 jest on 16-bitowy. Posiada jednak kilka dodatkowych funkcji które rozszerzają jego możliwości. Zaczniemy jednak po kolei.

Podobnie jak w licznikach opisanych wcześniej licznik T2 składa się z dwóch bajtów TH2 (starszy) i TL2 (młodszy). Po-



Rys. 3. Struktura liczników T0 i T1 w trybie 0 (a także w trybie 1)



Rys. 4. Struktura liczników T0 i T1 w trybie 2

dobnie jak T0 i T1 licznik T2 może pełnić rolę czasomierza, czyli zliczać impulsy wewnętrzne pochodzące z zegara procesora, może także zliczać impulsy zewnętrzne dzięki alternatywnej funkcji jednego z pinów portu P1 a mianowicie P1.0 – nóżka 1 procesora 8052.

Licznik ten posiada także możliwość automatycznego załadowania wartości początkowej określonej przez użytkownika a zapisanej w dwóch oddzielnych rejestrach 8-bitowych (które w sumie dają 16-bitową wartość początkową) zwanych RLDH i RLDL (patrz odcinek w EdW nr 6/97 tabela rejestrów specjalnych SFR).

Funkcja ta działa podobnie jak w liczniku T0 (T1) ustawionym w trybie 2. Załóżmy jednak że w przypadku T2 pracuje całe 16-bitów licznika i całe 16-bitów z RLDH.RLDL może być automatycznie załadowane, kiedy licznik zostanie przepelniony.

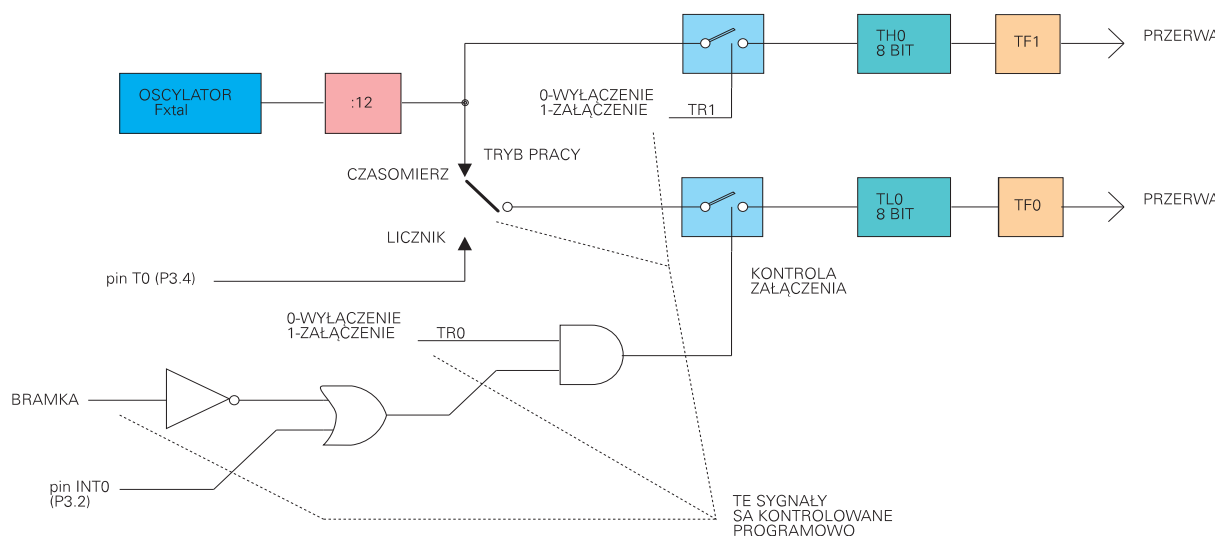
Z licznikiem T2 w kostce 80C52 związany jest dodatkowo także alternatywnie wykorzystywany pin portu P1 (zwany T2EX) a mianowicie P1.1 – nóżka 2. Otóż jeżeli zachodzi potrzeba, programista może wykorzystać tę końcówkę do zewnętrznego bramkowania licznika T2, co bardzo często przydaje się podobnie jak przy bramkowaniu liczników T0 i T1 sygnałami INT0 i INT1.

I tak w przypadku gdy T2 pracuje jako licznik liczący zewnętrzne impulsy, opadające zbrocze na końcówce T2EX spowoduje automatyczne natychmiastowe załadowanie licznika T2 – TH2 i TL2 wartością zdefiniowaną w rejestrach RLDH.RLDL. Do czego to wykorzystać? Jest wiele praktycznych zastosowań, ale chociażby funkcja autoladowania pod wpływem zewnętrznego sygnału może być przydatna do synchronizowania pracy wewnętrznych licznika z zewnętrznym sygnałem zegarowym o niższej częstotliwości.

Jeżeli zaś licznik T2 pracuje w roli czasomierza, to wejście T2EX można wykorzystać do automatycznego przepisania aktualnej wartości rejestrów TH2.TL2 do rejestrów RLDH.RLDL. Można powiedzieć że działanie w tym trybie (czasomierza) jest jakby odwrotne do sposobu w trybie licznika. Wartość zostaje przepisana do rejestrów RLDH.RLDL a nie odwrotnie jak to miało miejsce w przypadku pracy T2 w trybie licznika impulsów zewnętrznych.

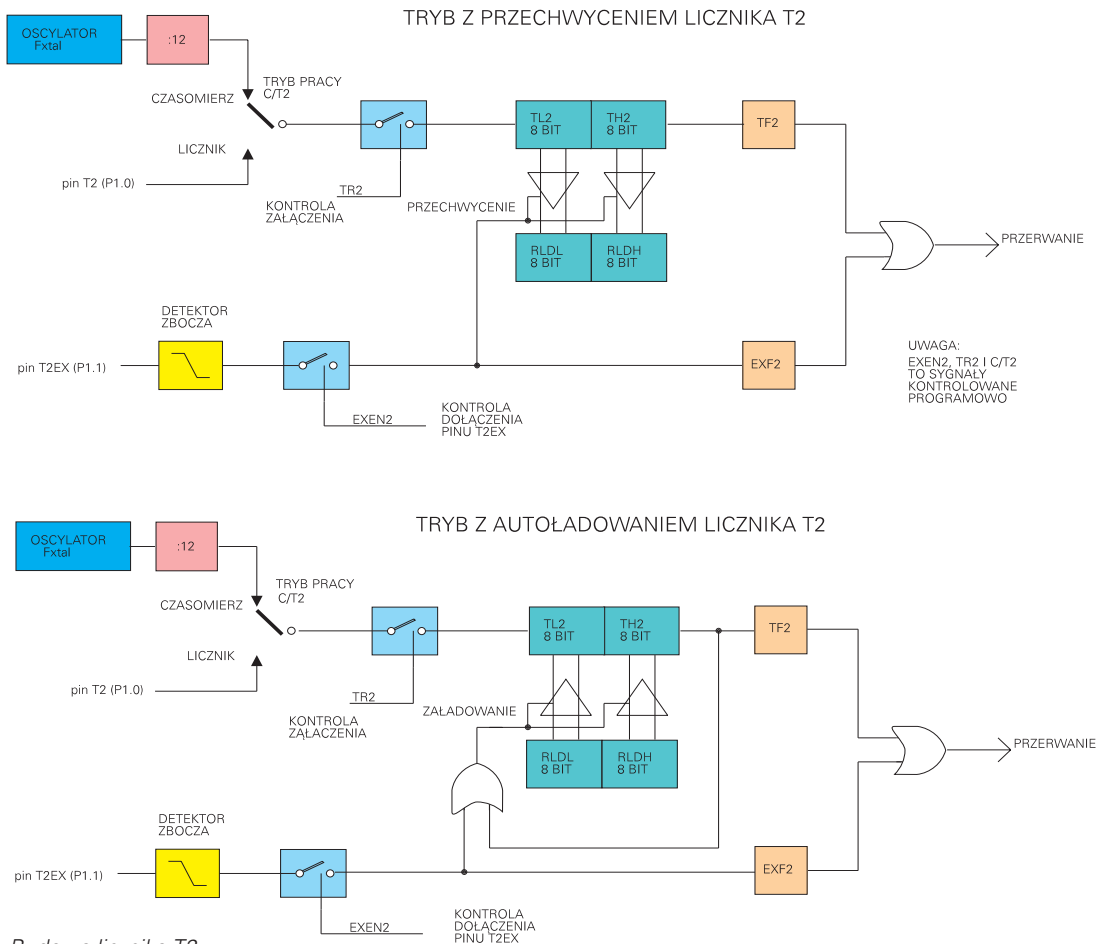
W praktyce takie działanie umożliwia np. na bardzo dokładny pomiar przebiegów wolnozmiennych bez konieczności stosowania dodatkowych układów scalonych. Odpowiedź na pytanie „A jak to się robi...?” otrzymasz drogi Czytelniku przy okazji kursu programowania 8051.

Na koniec istotna dodatkowa informacja dotycząca licznika T2. Podobnie jak licznik T1, T2 może w zależności od po-



Rys. 5. Struktura liczników T0 i T1 w trybie 3

Też to potrafisz



Rys. 6. Budowa licznika T2

trzeb, taktować port transmisji szeregowej. W takim przypadku możliwe jest rozszerzenie zakresów prędkości transmisji o dodatkowe wartości niedostępne przy tradycyjnym taktowaniu portu poprzez licznik T1, jak to opisano wcześniej.

Po tej porcji wiadomości, w następnym odcinku pozostanie nam do omó-

wienia układ transmisji szeregowej oraz układ przerwań. Te pozostałe dwa bloki funkcjonalne zakończą część zaznajamiającą Ciebie drogi Czytelniku z procesorami 8051 i 8052.

Wkrótce rozpoczniemy omówienie wszystkich rejestrów specjalnych, ale to już przy okazji pierwszych kroków w pro-

gramowaniu 8051. A propos,... czy zapoznałeś się już z układem „Komputerka edukacyjnego z 8051” opisanym w poprzednim i tym numerze EdW? Będzie on niezbędnym narzędziem podczas lekcji, toteż warto pomyśleć o jego zmontowaniu.

Sławomir Surowiński

Też to potrafisz ★

W kolejnym odcinku naszego cyklu, którego celem jest poznanie i nauczenie się programowania mikrokontrolerów serii MCS-51, postanowiłem odłożyć na bok pozostałe do omówienia bloki funkcjonalne procesora 8051 (port szeregowy, system przerwań, specjalne tryby pracy), a „wrzucić” Ci, drogi Czytelniku garść informacji pochodzącej trochę z „innej beczki”.

Chodzi mianowicie o krótkie, aczkolwiek wystarczające zapoznanie się z podstawowymi pojęciami dotyczącymi obsługi tych „wszystkich mądrości”, o których od kilku miesięcy uważnie czytasz – czyli o sposób programowania, czyli: „czym?”, „jak?”, i „dlaczego?... się ten procesor programuje”.

Decyzja moja jest po części Waszą, prawdopodobnie przeciw większości z Was ma już swój „Komputer edukacyjny”, którego konstrukcja została opisywana w dwóch poprzednich numerach EdW. A skoro wydałeś na to swoje oszczędności, to dobrze by było, nie patrzeć tylko na te „cudeńko” ale je w końcu wypróbować!



Do wspólnego zrealizowania tego zadania niezbędnych jest kilka informacji oraz zrozumienie pojęcia „programowania” układu scalonego – w naszym przypadku mikroprocesora.

Zanim przejdę do wyjaśnienia tych pojęć, pragnę uspokoić wszystkich drobiazgowych Czytelników, że wspomniane pozostałe bloki funkcjonalne procesora omówię w następnych odcinkach naszego cyklu, ale tym razem... uwaga: na gorąco, czyli z wykorzystaniem naszego edukacyjnego układu.

Tym wszystkim, którzy nie zaopatrzyli się w dedykowany temu kursowi, komputer, radzę o jak najszybsze zmontowanie go, dzięki czemu będziecie, moi drodzy, na bieżąco praktycznie wykonywać wszystkie zadania.

A więc zacznijmy od najważniejszego stwierdzenia: „Mikrokontroler bez programu jest jak żołnierz bez...” wiesz bez czego. I jest to święta racja. Jak sam zdążyłeś się zorientować śledząc poprzednie odcinki kursu, że kostka 8051 zawiera w swojej strukturze całe mnóstwo użytecznych elementów takich jak np. pamięć programu, danych, programowane układy licznikowe, stos, itd. Wszystko fajnie... „tylko jak nad tym wszystkim zapamiętać?...”. Otóż aby odpowiednio wykorzystać zasoby kontrolera i zmusić je do pracy zgodnie z naszym zamysłem i przeznaczeniem konstruowanego układu, po-

trzebny jest język porozumiewania się z mikroprocesorem. Ten język najczęściej nazywa się językiem maszynowym. Brzmi bardzo poważnie, prawda? Tak na prawdę język ten jest prostym zbiorem poleceń, dzięki którym możliwa jest nie tylko ingerencja we wszystkie wspomniane bloki funkcjonalne procesora, ale także wykonywanie określonych logicznych czynności: sprawdzanie warunków czy operacje arytmetyczno-logiczne. Sam język maszynowy to na pozór nieczytelny dla człowieka ciąg liczb. Aby sprawę uprościć stworzono postać jawną języka maszynowego – asembler. W języku tym kolejne polecenia opisywane są za pomocą słownych instrukcji uzupełnionych odpowiednimi do danej sytuacji argumentami. Tak postać jest akceptowalna przez programistę, dzięki temu program pisze się po prostu w dowolnym edytorze tekstowym (np. na komputerze), następnie dokonuje się zamiany (translacji) tak napisanego programu na wspomniany ciąg liczb – czyli język maszynowy procesora.

Tak jak istnieje na świecie wiele języków porozumiewania się między ludźmi, tak samo w rodzinach różnych mikroprocesorów, często pochodzących od różnych producentów, istnieje wiele języków, wszystkie jednak to języki maszynowe.

Producenci oprogramowania tworzą bardziej zaawansowane tzw. języki

„wyższego poziomu”, słyszałeś zapewne o takich jak Pascal, C, Basic oraz inne. Tak naprawdę to są to translatory bardziej złożonych poleceń danego języka „wyższego poziomu” na kod maszynowy procesora. Zawsze na samym końcu obróbki programu użytkownika, czy powstał on w takim czy innym języku powstaje i zawsze kod maszynowy, który akceptuje dedykowany mikroprocesor.

W przypadku pisania większości programów na kontrolery 8051 (lub każde inne „jednoukładowce”), szczególnie podczas nauki, każdy początkujący musi poznać jego język maszynowy, czyli asembler. Dzięki temu później, kiedy nauczy się nim biegle posługiwać, będzie mieć dużą swobodę i możliwość obiektywnej oceny w wyborze dowolnego innego narzędzia wspomagającego programowanie tego procesora, ale to zupełnie inna historia.

Ogólnie można powiedzieć, że pisanie programu na procesor to po prostu tworzenie kolejnych poleceń, z których w efekcie powstaje cały program. „Tworzenie” to np. pisanie w dowolnym edytorze tekstowym ASCII w przypadku posiadaczy komputerów. Pozostałe osoby nie mające dostępu, mogą taki program napisać chociażby na kartce papieru (aczkolwiek w przypadku większych programów jest to bardzo trudne, czy wręcz niemożliwe).

Polecenie zwykle zawiera się w jednej linii programu. Jego składnia jest zasadniczo określona, z reguły można ją przedstawić jako:

instrukcja <argument_1>, <argument_2>... (1)

Czasami przed instrukcją występuje także etykieta zakończona znakiem „:” (dwukropka), np. „etyk01:”, której zadaniem jest po prostu nazwanie danej linii polecenia. Ilość argumentów w linii polecenia może być różna w zależności od rodzaju instrukcji. W przypadku asemblera procesora 8051 i pochodnych liczba ta waha się od zera w przypadku instrukcji bezargumentowych do trzech. Zasada jest że poszczególne argumenty oddziela się zawsze znakiem „,”: (przecinka), natomiast instrukcja oddzielona jest od pierwszego argumentu spacją (odstępem) lub znakiem tabulacji (to informacja dla komputerowców).

Podajmy przykład polecenia dla procesora 8051 w którym procesor wykonuje dodawanie zawartości dwóch jego rejestrów:

ADD A, B (2)

W poleceniu tym wystąpiły:

- instrukcja: „ADD”
- argumenty: A – rejestr A (akumulator) oraz rejestr B

W nazewnictwie asemblerowym pierwszą część polecenia, czyli instrukcję (np. ADD) nazywa się „mnemonikiem”, toteż od tego momentu będziemy posługiwać się tym zwrotem. Aby otrzymać wynik dodawania należy dodać do siebie dwa argumenty. Pierwszy argument (jakaś liczba) znajduje się w akumulatorze procesora 8051, druga zaś w tym przypadku w rejestrze B. W wyniku wykonania przytoczonego polecenia procesor doda do zawartości rejestru A wartość z rejestru B, a wynik umieści w rej. A (akumulatorze). Powiedzmy że chcemy dodać dwie liczby: 25 + 43. W tym celu musimy wpisać te wartości (składniki dodawania) do rejestrów procesora a potem jej dodać poleceniem (2). Można to zrobić w sposób następujący:

MOV A, #25 (3)

MOV B, #43 (4)

ADD A, B (5)

W liniach 3 i 4 rozkazaliśmy procesorowi wpisanie składników odpowiednio do rejestrów A i B, działanie polecenia w linii (5) jest Ci już znane.

Osoby znające asembler i możliwości 8051 pewnie się trochę zaśmieją z przykładu (3)...(5), bowiem dodawanie 2 stałych da się zapisać krócej w dwóch liniach, lecz nie o to nam chodzi, przynajmniej na tym etapie kursu.

Mnemonik „MOV” ma wiele zastosowań w języku procesorów '51, na język polski można ją przetłumaczyć jako „przemieszczenie” (przesunięcie) czegoś

gdzieś. Czego i gdzie to zależy od kontekstu, czyli od rodzaju argumentów, ale o tym za chwilę. Linia (3) oznacza dosłownie w tłumaczeniu na jęz. polski: „wpisz do akumulatora liczbę 25”, linia (4) – wpisz do rejestru B liczbę 43, linia (5) – dodaj do akumulatora zawartość rejestru B. W efekcie wykonania tych 3 poleceń w akumulatorze będzie wynik dodawania, czyli liczba 68 (binarnie będzie to 01000100). Jeżeli teraz każesz procesorowi wykonać np. polecenie

MOV P1, A (6)

spowoduje to że liczba 68 pojawi się na końcówkach portu P1 procesora, oczywiście w postaci binarnej. Aby to sprawdzić naocznie wystarczy dołączyć do każdej z 8-miu jego końcówek (piny 1...8 '8051/2) diodę świecącą z rezystorem, z pewnością zapalone zostaną diody dołączające do końcówek: 3 i 7 (na tych pozycjach liczby 68 w postaci binarnej jest „1”). Procesor po prostu w wyniku podania polecenia (6) wpisał zawartość akumulatora do rejestru portu P1.

No ale na razie wystarczy przykładów, wyjaśnimy sobie teraz, jak fizycznie procesor akceptuje instrukcje, no bo przecież „nie zje” od razu ciągu liter układających się w napis chociażby z przykładu (6). Otóż asembler procesora 8051 posiada określoną liczbę mnemoników oraz określone rodzaje argumentów.

Jeżeli jesteś ciekawy to powiem Ci że mnemoników jest nie tak wiele bo 42. W przyszłości będziesz musiał je poznać i zapamiętać, ale nie przejmuj się, nie jest to dużo, przy okazji praktycznego programowania same wpadną ci do głowy – i już zostaną...na zawsze.

Jeżeli chodzi o argumenty to wyróżnia się kilka ich rodzajów, nie będę teraz szczegółowo wyliczał ich wszystkich, pierwszy ich rodzaj – argument bezpośredni, poznałeś w przykładzie dodawania dwóch rejestrów – liczby: #25 i #43.

W zależności od rodzaju argumentów jakie występują po mnemoniku, różne jest działanie całego polecenia – instrukcji. W architekturze procesorów 8051 konstruktorzy wyróżnili 255 takich sytu-

cji i ponumerowali je od 0 do 255 (FFh szesnastkowo).

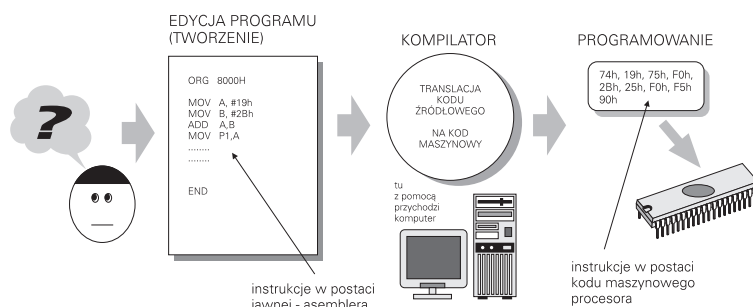
Tak powstało 255 instrukcji procesora (nie 256 bo jeden numer nie jest wykorzystany).

I jak się pewnie niedługo przekonasz z punktu widzenia programisty – użytkownika liczba ta jest mniejsza, to jednak należy zapamiętać ten fakt.

Jeżeli zatem cały zbiór instrukcji procesora daje się przedstawić jako liczba wraz z towarzyszącymi jej ewentualnie argumentami, z których każdy także daje się przetłumaczyć na liczbę, w efekcie można wywnioskować, że cały program pisany przez użytkownika w postaci źródłowej (literowej) można przetłumaczyć na ciąg liczb. Mało tego, wszystkie te liczby mogą zawierać się jedynie w 8 bitach, co idealnie pasuje architekturze naszego procesora – jest on przecież prawdziwym ośmiobitowcem. Zamieniony do takiej postaci Twój program wystarczy teraz śmiało wpisać do poszczególnych komórek pamięci EPROM bądź samego procesora (gdy ten pracuje w trybie z wewnętrzzną pamięcią programu – np. 87C51) lub do kostki EPROM z której później procesor będzie pobierał instrukcje.

Operację programowania pamięci EPROM przeprowadza się oczywiście za pomocą specjalizowanych narzędzi do programowania procesorów – tzw. programatorów. Na pocieszenie powiem Ci że w ofercie handlowej AVT pod nazwą AVT-320 znajduje się taki programator idealnie nadający się do programowania wszystkich dostępnych na rynku kostek serii MCS-51. W przyszłości, kiedy nabydziesz już umiejętności swobodnego „surfowania” (to ostatnio bardzo popularne słowo) wśród rodziny '51-nek, z pewnością takie narzędzie Ci się przyda. To przyszłość, na razie do tego etapu jeszcze wspólnie nie doszliśmy.

Całą drogę od pomysłu na program poprzez jego napisanie i zamianę do postaci akceptowanej przez procesor i jednocześnie dającej wpisać się do pamięci programu procesora obrazuje **rysunek 1**.



Rys. 1. Od pomysłu na program do jego realizacji

Też to potrafisz

Wróćmy na chwilę do naszego prostego przykładu dodawania dwóch liczb. Zapiszmy instrukcje (3)...(5) z argumentami dodawania w postaci szesnastkowej:

```
MOV A, #19h (7)
MOV B, #2Bh (8)
ADD A, B (9)
```

Liczba 25 dziesiętnie można zapisać w postaci heksadecymalnej jako „19h”, a liczba 43 jako „2Bh”, dodając na końcu każdej z nich małą literkę „h” co oznacza zapis że zapisaliśmy liczbę w takiej właśnie postaci. Spróbujmy teraz zamienić te trzy linie na postać liczbową (bajtową) akceptowaną przez procesor. W liście rozkazów 8051 instrukcja:

```
MOV A, "jakaś_liczba_8_bitowa"
```

ma numer (odtąd będziemy ten numer nazywać kodem rozkazu): „74h”. Ale w linii (7) występuje jeszcze argument – liczba stała „19h”, dlatego ostatecznie linię tę w kodzie maszynowym (liczbowo) można zapisać jako: „74h, 19h”. Podobnie tłumaczymy linię (8). Odwołując się do listy instrukcji (którą całą niebawem poznasz), sekwencję:

```
MOV B, "jakaś_liczba_8_bitowa"
```

tłumaczymy jako: „75h, F0h, 2Bh”. Trzecia linia zaś będzie miała postać: „25h, F0h”. Skąd to wszystko wiem? Ano ze wspomnianej listy instrukcji. Nie martw się w tej chwili nie jest Ci ona potrzebna, ważne jest abyś uzmysłowił sobie w jaki sposób pisze się program w języku assemblera i jak go potem tłumaczy się na język maszynowy procesora – czyli postać liczbową.

W efekcie po przetłumaczeniu naszego przykładu otrzymamy sekwencję liczb: „74h, 19h, 75h, F0h, 2Bh, 25h, F0h „ (10)

Jeżeli teraz poszczególne liczby wpiszesz do kolejnych komórek pamięci programu (czy to zewnętrznej czy to wewnętrznej) to po uruchomieniu układu procesor wykona dokładnie to czego do niego oczekujesz, czyli załaduje dwa składniki do dwóch rejestrów procesora a następnie dokona operacji dodania ich.

Uff, wyglądało to dość mozolnie, bo trzeba było napisać instrukcje w postaci assemblerowej czyli jawnej (linie 7,8,9), potem na podstawie bliżej Ci nie znanej (na razie) tabeli instrukcji zamienić program do postaci maszynowej, wreszcie umieścić sekwencję w pamięci programu.

W praktyce dzięki zastosowaniu dowolnego komputera proces ten da się przyspieszyć.

O ile każdy program w postaci assemblera trzeba wstukać z klawiatury i zapisać na dysku w postaci pliku tekstowego (np. korzystając z edytora popularnego Norton Commandera), to do przetłumaczenia postaci źródłowej za wykonywalną (maszynową) służą specjalne narzędzia

(programy) zwane kompilatorami. Dzięki nim proces zamiany – zwany dalej kompilacją – kodu źródłowego na maszynowy trwa często bardzo krótko, a tłumaczenie nawet kilku tysięcy linii nie trwa dłużej niż kilkanaście sekund. W efekcie działania kompilatora powstaje zbiór z programem zapisany w postaci maszynowej, czyli ciągu liczb jak zilustrowałem na przykładzie dodawania liczb. Plik taki najczęściej jest gotowy do użycia przez programatory pamięci EPROM (lub programatory procesorów). Taki zbiór świetnie nadaje się też do zapisania w pamięci operacyjnej twojego komputera edukacyjnego opisywanego w trzech ostatnich numerach EdW. Jeżeli posiadasz dowolny komputer klasy PC, będziesz mógł nabyć dyskietkę z takim kompilatorem, dzięki któremu efekty twojej pracy będą natychmiastowe. Szczegółowe informacje zawarte są w 3 części opisującej komputer edukacyjny na 8051 w tym numerze EdW.

Jeżeli nie masz dostępu do komputera, będziesz zmuszony do tłumaczenia przykładów z naszego kursu, ręcznie na kartce papieru, korzystając ze „ściągawki”, którą będzie lista instrukcji procesora 8051. Lista taka ukaże się

w przyszłym numerze EdW. Będziesz ją mógł wyciąć i w razie potrzeby zafolować, chroniąc ją tym samym przed zniszczeniem.

I choć wszystkie przykłady w czasie programowania będą dość proste i dające się zrealizować „na papierze”, to powinienś już teraz pomyśleć o wyposażeniu swego domowego kącika, nawet w przestarzały komputer klasy AT-286 lub w muzealną wersję XT.

Użytkownicy komputerów innych rodzajów, np. Amiga, Commodore, Atari, posiadający interfejs szeregowy zgodny z RS232c będą mogli także ich używać, do przesyłania kodu maszynowego pisanych przez siebie programów z komputera do naszego systemiku edukacyjnego. Musicie jednak kochani poszperać wśród swoich kolegów i namierzyć kompilator na procesory 8051 działający na waszym komputerze, bowiem na dyskietce oferowanej do naszego kursu znajduje się zestaw programów na komputery klasy PC.

W tym odcinku dość nietypowo, umieszczamy pierwsze 5 ćwiczeń w części III opisu zestawu AVT-2250. Zapraszam do lektury w tym numerze EdW.

Sławomir Surowiński

Też to potrafisz ★

W kolejnym odcinku poświęconym naszym wspólnym staraniom mającym na celu ujarzmienie mikrokontrolera 8051 postaram się zapoznać Was drodzy Czytelnicy w przystępny sposób z listą instrukcji tego procesora. Na końcu tego odcinka czeka na Was druga już lekcja – czyli kolejny praktyczny krok w nauce z wykorzystaniem naszego komputera edukacyjnego z 8051. Dziś wspólnie napiszemy i przeanalizujemy krótki ale już prawdziwie asemblerowy program



W poprzednim odcinku poznałeś już ideę tworzenia programów na mikrokontrolery 8051. Wiesz że do tego celu niezbędny jest zestaw instrukcji danego procesora (u nas jest to rodzina MCS-51, która ma wspólny język programowania) oraz znajomość kodów numerycznych poszczególnych instrukcji w przypadku kiedy nie masz dostępu do komputera i wszystkie czynności musisz wykonać ręcznie. W przypadku kiedy do dyspozycji programisty jest komputer, procedurę tłumaczenia instrukcji zapisanych jawnie – w języku asemblera – automatycznie wykonuje komputer korzystający z programu zwanego kompilatorem. Autor cyklu zadbał, aby każdy z Was drodzy Czytelnicy, niskim kosztem mógł stać się posiadaczem takiego kompilatora. Jest on dostępny na dyskietkach 3,5" w ofercie handlowej AVT pod nazwą AVT-2250/D. Ważną informacją jest to że zamieszczono tam dwie wersje kompilatora: wersję angielską oraz polską!. Jest to chyba pierwszy program tego typu komunikujący się w naszym ojczystym języku z programistą. Dzięki temu osoby nie znające angielskiego będą mogły bez problemów korzystać z takiej wersji kompilatora. Funkcjonalnie obie wersje są takie same, to znaczy że wykonują wszystkie czynności identycznie, jedynie komunikaty zgłaszane przez program występują w dwóch różnych językach, jak wspominałem wcześniej. Na dyskietce

znajduje się plik tekstowy z rozszerzeniem .DOC, w którym zawarte są informacje o kompilatorze i jego możliwościach niezbędne do prawidłowego posługiwania się nim. Dlatego nie zbędę opisywać szczegółowo tych spraw, ponieważ wśród naszych czytelników są osoby nie mające komputera PC a poza tym każdy zainteresowany PC-towiec będzie miał sam okazję na zapoznanie się z instrukcją użytkownika programu.

Ze względu na mocno ograniczone możliwości „ręcznej” kompilacji stworzonych przez Ciebie programów do postaci maszynowej, powinieneś już teraz zastanowić się nad możliwością nabycia lub przynajmniej korzystania z komputera PC, nawet pocziwej AT czy XT. Efektywne, pozbawione niepotrzebnych pomyłek, tworzenie nawet mało skomplikowanych programów jest możliwe tylko przy wykorzystaniu komputera oraz kompilatora, który jest dostępny dla wszystkich zainteresowanych po przystępnej cenie.

W tym miejscu chcę uspokoić wszystkich antykomputerowców. Wszystkie przedstawiane w cyklu przykładowe programy będą drukowane w postaci czytelnej i jasnej także dla tego grona czytelników. Ułatwi to analizę i pokaże jak w praktyce tłumaczyć się komendy asemblera na język maszynowy.

W tym odcinku szkoły mikroprocesorowej zapoznamy się z listą instrukcji procesora,

oraz dodatkowo zbierzemy „w całość” wiadomości dotyczące wszystkich rejestrów specjalnych SFR – także tych nie omawianych (na razie). Wszystko to jest umieszczone dodatkowo we wkładce wewnątrz numeru w postaci kartki A4 z nadrukowanymi dwustronnie skrótkowo wszystkimi informacjami niezbędnymi do rozpoczęcie pisanie własnych programów oraz ich tłumaczenia (asemblacji) w przypadku osób które muszą to zrobić ręcznie. Taka „ściąga” powinna być przez Ciebie drogą Czytelniku wycięta a następnie zafoliowana, by mogła ci służyć przez cały czas zabawy z procesorem 8051. Zawieszenie jej na ścianie nad twoim biurkiem z pewnością ułatwi Ci poznanie i zapamiętanie wszystkich instrukcji procesora, tak abyś w przyszłości mógł władać asemblerem tak ja własnym ojczystym językiem – gwarantuję Ci – jest to możliwe!

Przejdźmy zatem do zapoznania się i wyjaśnienia działania wszystkich poleceń kontrolerów 8051.

Lista instrukcji

Informacje zawarte w tej części artykułu są rozszerzeniem listy przedstawionej we wkładce wewnątrz numeru. Dlatego analizują opisy poszczególnych instrukcji powinieneś mieć także przed oczyma wspomnianą „ściągę”. Kiedy w przyszłości nabierzesz nieco wprawy w posługiwaniu się poleceniami asemblera, potrzebna będzie Ci tylko strona z wkładki,

a do niniejszego opisu będzie mógł zawsze wrócić w przypadku niejasności, szczególnie wtedy jeżeli będziesz chciał tworzyć programy nie mając dostępu do komputera PC. Tak więc zaczynamy.

Opis każdej instrukcji składa się zasadniczo z następujących elementów:

- nazwy angielskiej i polskiej instrukcji: pkt. a)
- krótkiego opisu działania instrukcji: pkt. b)
- wyszczególnienia znaczników na które działa instrukcja: pkt. c)
- opisu szczegółowego instrukcji lub jej rodzajów: pkt. d), wraz z podaniem formatu i kodów maszynowych instrukcji, w zapisanych binarnie i heksadecymalnie, z podaniem ilości cykli i bajtów kodu oraz ewentualnie poparte przykładem lub uwagami dotyczącymi efektów użycia danej instrukcji.

Większość z tych informacji znajduje się także w tabeli zestawieniowej instrukcji we wkładce wewnątrz numeru.

Operacje arytmetyczne

"ADD"

- a) ang. „add to accumulator” – dodaj do akumulatora
- b) Do wartości przechowywanej w akumulatorze dodawany jest wskazany argument, a wynik zostaje wpisany do akumulatora.
- c) znaczniki: C, AC i OV
- d) rodzaje instrukcji:

– ADD A, Rn

do akumulatora dodawana jest zawartość rejestru Rn
 $A \leftarrow A + Rn$ gdzie Rn = R0...R7 (jeden z rejestrów roboczych)
 kod: 0 0 1 0 1 n2 n1 n0 gdzie n2...n0 – wskazują na R0...7 stąd: 28h-2Fh
 cykl: 1 bajty: 1
 przykład: ADD A, R2

– ADD A, adres

do akumulatora dodawana jest zawartość komórki w wewn. RAM o adresie „adres”
 $A \leftarrow A + (\text{adres})$
 kod: 0 0 1 0 0 1 0 1 25h
 cykl: 1 bajty: 2 (kod instrukcji 25h + adres)
 przykład: ADD A, 2Fh (dodanie do A zawartości komórki o adresie 2Fh)

– ADD A, @Ri

do akumulatora dodawana jest zawartość komórki w wewn. RAM o adresie wskazywanym przez rejestr Ri (R0 lub R1)
 $A \leftarrow A + (Ri)$
 kod: 0 0 1 0 0 1 1 i gdzie i wskazuje na R0 (i=0) lub R1 (i=1) stąd: 26h, 27h
 cykl: 1 bajty: 1
 przykład: ADD A, @R0 (dodanie do A zawartości komórki o adresie w R0)

– ADD A, #dana

do akumulatora dodawany jest argument stały (8-bitowa liczba)
 $A \leftarrow A + \text{dana}$
 kod: 0 0 1 0 0 1 0 0
 cykl: 1 bajty: 2 (kod instrukcji + dana)
 przykład: ADD A, #120 (dodanie do A liczby 120)

"ADDC"

- a) ang. „add to accumulator with carry” – dodaj do akumulatora z przeniesieniem

- b) Do wartości przechowywanej w akumulatorze dodawany jest wskazany argument oraz zawartość znacznika przeniesienia C, a wynik zostaje wpisany do akumulatora.

- c) znaczniki: C, AC i OV

- d) rodzaje instrukcji:

– ADDC A, Rn

do akumulatora dodawana jest zawartość rejestru Rn oraz C
 $A \leftarrow A + Rn + C$ gdzie Rn = R0...R7 (jeden z rejestrów roboczych)
 kod: 0 0 1 1 1 n2 n1 n0 gdzie n2...n0 – wskazują na R0...7 stąd: 38h-3Fh
 cykl: 1 bajty: 1
 przykład: ADDC A, R4

– ADDC A, adres

do akumulatora dodawana jest zawartość komórki w wewn. RAM o adresie „adres” oraz znacznik C
 $A \leftarrow A + (\text{adres}) + C$
 kod: 0 0 1 1 0 1 0 1 35h
 cykl: 1 bajty: 2 (kod instrukcji 35h + adres)
 przykład: ADDC A, 7Eh (dodanie do A zawartości komórki o adresie 7Eh)

– ADDC A, @Ri

do akumulatora dodawana jest zawartość komórki w wewn. RAM o adresie wskazywanym przez rejestr Ri (R0 lub R1) oraz C
 $A \leftarrow A + (Ri) + C$
 kod: 0 0 1 1 0 1 1 i gdzie i wskazuje na R0 (i=0) lub R1 (i=1) stąd: 36h, 37h
 cykl: 1 bajty: 1
 przykład: ADDC A, @R1 (dodanie do A zawartości komórki o adresie w R1)

– ADDC A, #dana

do akumulatora dodawany jest argument stały (8-bitowa liczba) i C
 $A \leftarrow A + \text{dana} + C$
 kod: 0 0 1 1 0 1 0 0
 cykl: 1 bajty: 2 (kod instrukcji + dana)
 przykład: ADDC A, #120 (dodanie do A liczby 120)

"SUBB"

- a) ang. „subtract from accumulator with borrow” – odejmij od akumulatora z pożyczką
- b) Od wartości przechowywanej w akumulatorze odejmowany jest wskazany argument oraz zawartość znacznika przeniesienia C, a wynik zostaje wpisany do akumulatora.

- c) znaczniki: C, AC i OV

- d) rodzaje instrukcji:

– SUBB A, Rn

od akumulatora odejmowana jest zawartość rejestru Rn oraz C
 $A \leftarrow A - Rn - C$ gdzie Rn = R0...R7 (jeden z rejestrów roboczych)
 kod: 1 0 0 1 1 n2 n1 n0 gdzie n2...n0 – wskazują na R0...7 stąd: 98h-9Fh
 cykl: 1 bajty: 1
 przykład: SUBB A, R6

– SUBB A, adres

od akumulatora odejmowana jest zawartość komórki w wewn. RAM o adresie „adres” oraz znacznik C
 $A \leftarrow A - (\text{adres}) - C$
 kod: 1 0 0 1 0 1 0 1 95h
 cykl: 1 bajty: 2 (kod instrukcji 95h + adres)
 przykład: SUBB A, 45h (odjęcie od A zawartości komórki o adresie 45h i znacznika C)

– SUBB A, @Ri

od akumulatora odejmowana jest zawartość komórki w wewn. RAM o adresie wskazywanym przez rejestr Ri (R0 lub R1) oraz C

$$A \leftarrow A - (Ri) - C$$

kod: 1 0 0 1 0 1 1 i gdzie i wskazuje na R0 (i=0) lub R1 (i=1) stąd: 96h, 97h

cykl: 1 bajty: 1

przykład: SUBB A, @R1 (odjęcie od A zawartości komórki o adresie w R1 oraz C)

– SUBB A, #dana

od akumulatora odejmowany jest argument stały (8-bitowa liczba) oraz C
 $A \leftarrow A - \text{dana} - C$
 kod: 1 0 0 1 0 1 0 0 94h
 cykl: 1 bajty: 2 (kod instrukcji + dana)
 przykład: ADDC A, #86h (odjęcie od A liczby 86h i znacznika C)

"INC"

- a) ang. „increment” – zwiększenie o 1
- b) do wskazanego argumentu jest dodawana jedynka

- c) znaczniki: bez zmian

- d) rodzaje instrukcji:

– INC A

do akumulatora dodawana jest jedynka
 $A \leftarrow A + 1$
 kod: 0 0 0 0 0 1 0 0 04h
 cykl: 1 bajty: 1

– INC Rn

do zawartości rejestru Rn dodawana jest jedynka
 $Rn \leftarrow Rn + 1$ gdzie Rn = R0...R7 (jeden z rejestrów roboczych)
 kod: 0 0 0 0 1 n2 n1 n0 gdzie n2...n0 – wskazują na R0...7 stąd: 08h-0Fh
 cykl: 1 bajty: 1
 przykład: INC R3

– INC adres

do zawartości komórki o adresie „adres” dodawana jest jedynka
 $(\text{adres}) \leftarrow (\text{adres}) + 1$
 kod: 0 0 0 0 0 1 0 1 05h
 cykl: 1 bajty: 2 (kod instrukcji 05h + adres)
 przykład: INC 12h (inkrementacja zawartości komórki o adresie 12h)

– INC @Ri

do zawartości komórki o adresie wskazywanym przez Ri dodawana jest jedynka
 $(Ri) \leftarrow (Ri) + 1$
 kod: 0 0 0 0 0 1 1 i gdzie i wskazuje na R0 (i=0) lub R1 (i=1) stąd: 06h, 07h
 cykl: 1 bajty: 1
 przykład: INC @R1

– INC DPTR

do 16-bitowego wskaźnika danych DPTR złożonego z rejestrów SFR: DPH (bardziej znaczący bajt) i DPL (mniej znaczący bajt) dodawana jest jedynka.
 Znaczniki nie ulegają zmianie.
 $DPTR \leftarrow DPTR + 1$
 kod: 1 0 1 0 0 0 1 1 A3h
 cykl: 2 bajty: 1

"DEC"

- a) ang. „decrement” – zmniejszenie o 1
- b) od wskazanego argumentu odejmowana jest jedynka

- c) znaczniki: bez zmian

- d) rodzaje instrukcji:

– DEC A

od akumulatora odejmowana jest jedynka
 $A \leftarrow A - 1$
 kod: 0 0 0 1 0 1 0 0 14h
 cykl: 1 bajty: 1

– DEC Rn

od zawartości rejestru Rn odejmowana jest jedynka
 $Rn \leftarrow Rn - 1$ gdzie Rn = R0...R7 (jeden z rejestrów roboczych)

Też to potrafisz

kod: 0 0 0 1 1 n2 n1 n0 gdzie n2...n0 –
wskazują na R0...7 stąd: 18h-1Fh
cykle: 1 bajty: 1
przykład: DEC R5

– DEC adres

od zawartości komórki o adresie „adres”
odejmowana jest jedynka
(adres) <- (adres) - 1
kod: 0 0 0 1 0 1 0 1 15h
cykle: 1
bajty: 2 (kod instrukcji 15h + adres)
przykład: DEC 3Fh (inkrementacja zawar-
tości komórki o adresie 3Fh)

– DEC @Ri

od zawartości komórki o adresie wskazy-
wanym przez Ri odejmowana jest jedynka
(Ri) <- (Ri) - 1
kod: 0 0 0 1 0 1 1 i gdzie i wskazuje na
R0 (i=0) lub R1 (i=1) stąd: 16h, 17h
cykle: 1 bajty: 1
przykład: DEC @R0

“MUL AB”

- a) ang. „multiply” – pomnóż
- b) 8-bitowa liczba bez znaku znajdująca się w aku-
mulatorze jest mnożona przez 8-bitową liczbę
bez znaku z rejestru B. 16-bitowy wynik wpisy-
wany jest do rejestrów B i A (bardziej znaczący
bajt do B, mniej znaczący bajt do A)
- c) znaczniki: jeśli wynik mnożenia jest > 255 to
ustawiany jest znacznik OV, w przeciwnym razie
OV jest zerowany. znacznik C jest zerowany.
- d) B.A <- A x B
kod: 1 0 1 0 0 1 0 0 A4h
cykle: 4 bajty: 1

“DIV AB”

- a) ang. „divide” – podzieli
- b) 8-bitowa liczba bez znaku, znajdująca się
w akumulatorze jest dzielona przez 8-bito-
wą liczbę z rejestru B. Część całkowita ilora-
zu wypisywana jest do akumulatora, a reszta
z dzielenia do rejestru B. W przypadku gdy
dzielnik jest równy 0 (B=0) to po wykonaniu
operacji zawartość akumulatora i rejestru
B jest nieokreślona oraz dodatkowo usta-
wiony zostaje znacznik OV.
- c) znaczniki: C = 0, OV = 0 (zerowane)
- d) A <- A : B B <- reszta (A : B)
kod: 1 0 0 0 0 1 0 0 84h
cykle: 4 bajty: 1

“DA A”

- a) ang. „decimal adjust” – korekcja dziesiętna
- b) wykonywana jest korekcja dziesiętna wyni-
ku dodawania. Operacja ta sprowadza wynik
do postaci dwóch cyfr dziesiętnych w kodzie
BCD, jeżeli argumenty były w kodzie BCD.
Rozkaz ten powinien być używany jedynie
w połączeniu z rozkazem dodawania (ADD,
ADDC). Także inkrementacja powinna odby-
wać się poprzez instrukcję ADD A, #1, a nie
INC A, bowiem w tym drugim przypadku nie
są ustawianie znaczniki C i AC, tak więc nie
może być wykonana korekcja dziesiętna.
Korekcja polega na tym że w przypadku kiedy
po wykonanej na akumulatorze operacji doda-
wania (ADD, ADC) zawartość jego bitów
3...0 jest większa od 9 lub jest ustawiony
znacznik AC, to do wartości akumulatora doda-
wana jest liczba 6. Po tym jeżeli okaże się że
zawartość bitów 7...4 jest większa od 9 lub jest
ustawiony znacznik C to do tych bitów doda-
wana jest także liczba 6. Jeżeli podczas tej
ostatniej operacji wystąpiło przeniesienie to do
znacznika wpisywana jest 1, w przeciwnym
wypadku stan znacznika C nie zmienia się.
- c) znaczniki: C, OV (patrz pkt.a)
- d) A <- korekcja dziesiętna (A)
kod: 1 1 0 1 0 1 0 0 D4h

cykle: 1 bajty: 1
Przykład: jeżeli w wyniku dodawania
w akumulatorze jest liczba 6Ah, to po ko-
rekcji dziesiętnej akumulator będzie zawie-
rał liczbę 70h, patrz listing:
MOV A,#69h ;wpisanie liczby 69h
do akumulatora (1)
ADD A,#1 ;inkrementacja
akumulatora poprzez
dodawanie (2)
;w wyniku tego
w A będzie liczba 6Ah
da A ;korekcja dziesiętna A,
w A będzie po tym
70h (3).

Uwaga: jeżeli w przykładzie w linii (2) użyje-
my instrukcji INC A zamiast dodania jedynki,
to korekcja dziesiętna będzie nieprawidłowa.

Operacje logiczne

“ANL”

- a) ang. „logical AND” – pomnóż logicznie
- b) wykonywany jest iloczyn logiczny AND
(mnożenie bitów „bit po bicie”) wskaza-
nych w instrukcji dwóch argumentów. Wy-
nik operacji jest wpisywany do argumentu
pierwszego instrukcji
- c) znaczniki: nie zmieniają się
- d) rodzaje instrukcji:
 - ANL A, Rn
wymnożona logicznie zostaje zawartość
akumulatora i rejestru Rn, wynik w A
A <- A ∩ Rn gdzie Rn = R0...R7 (jeden
z rejestrów roboczych)
kod: 0 1 0 1 1 n2 n1 n0 gdzie n2...n0 –
wskazują na R0...7 stąd: 58h-5Fh
cykle: 1 bajty: 1
przykład: ANL A, R3

– ANL A, adres

wymnożona logicznie zostaje zawartość
akumulatora i komórki o podanym adresie
„adres”, wynik zostaje umieszczony w A
A <- A ∩ (adres)
kod: 0 1 0 1 0 1 0 1 55h
cykle: 1
bajty: 2 (kod instrukcji 55h + adres)
przykład: ANL A, 45h (mnożenie logiczne
A i zawartości komórki pod adresem 45h)

– ANL A, @Ri

wymnożona logicznie zostaje zawartość aku-
mulatora i komórki wewn. RAM o adresie
wskazywanym przez rejestr Ri. (R0 lub R1)
A <- A ∩ (Ri)
kod: 0 1 0 1 0 1 1 i gdzie i wskazuje na R0
(i=0) lub R1 (i=1) stąd: 56h, 57h
cykle: 1 bajty: 1
przykład: ANL A, @R1 (wymnożenie logicz-
ne A i zawartości komórki o adresie w R1)

– ANL A, #dana

wymnożona logicznie zostaje zawartość aku-
mulatora przez argument stały (8-bitowa liczba)
A <- A ∩ dana
kod: 0 1 0 1 0 1 0 0 54h
cykle: 1
bajty: 2 (kod instrukcji 54h + dana)
przykład: ANL A, #23 (mnożenie logiczne
A i liczby 23)

– ANL adres, A

wymnożona logicznie zostaje zawartość
akumulatora i komórki o podanym adre-
sie „adres”, wynik zostaje umieszczony
w komórce pamięci o adresie „adres”
(adres) <- (adres) ∩ A
kod: 0 1 0 1 0 0 1 1 52h
cykle: 1 bajty: 2 (kod instrukcji
55h + adres)
przykład: ANL A, 45h (mnożenie logiczne
A i zawartości komórki pod adresem 45h)

– ANL adres, #dana

wymnożona logicznie zostaje zawartość
komórki o adresie „adres” przez argument
stały (8-bitowa liczba)
(adres) <- (adres) ∩ dana
kod: 0 1 0 1 0 0 1 1 53h
cykle: 2 bajty: 3 (kod instrukcji
54h + adres + dana)
przykład: ANL 45h, #23 (mnożenie logicz-
ne zawartości komórki 45h i liczby 23)

“ORL”

- a) ang. „logical OR” – zsumuj logicznie
- b) wykonywana jest suma logiczna OR (doda-
wanie bitów „bit po bicie”) wskazanych
w instrukcji dwóch argumentów. Wynik
operacji jest wpisywany do argumentu pier-
wszego instrukcji
- c) znaczniki: nie zmieniają się
- d) rodzaje instrukcji:

– ORL A, Rn

dodana logicznie zostaje zawartość akumu-
latora i rejestru Rn, wynik w A
A <- A ∪ Rn gdzie Rn = R0...R7 (jeden
z rejestrów roboczych)
kod: 0 1 0 0 1 n2 n1 n0 gdzie n2...n0 –
wskazują na R0...7 stąd: 48h-4Fh
cykle: 1 bajty: 1
przykład: ORL A, R7

– ORL A, adres

dodana logicznie zostaje zawartość akumu-
latora i komórki o podanym adresie
„adres”, wynik zostaje umieszczony w A
A <- A ∪ (adres)
kod: 0 1 0 0 0 1 0 1 45h
cykle: 1 bajty: 2 (kod instrukcji
45h + adres)
przykład: ORL A, 19h (dodanie logiczne
A i zawartości komórki pod adresem 19h)

– ORL A, @Ri

dodana logicznie zostaje zawartość akumu-
latora i komórki wewn. RAM o adresie
wskazywanym przez rejestr Ri. (R0 lub R1)
A <- A ∪ (Ri)
kod: 0 1 0 0 0 1 1 i gdzie i wskazu-
je na R0 (i=0) lub R1 (i=1) stąd: 46h, 47h
cykle: 1 bajty: 1
przykład: ORL A, @R0 (dodanie logiczne
A i zawartości komórki o adresie w R0)

– ORL A, #dana

dodana logicznie zostaje zawartość akumu-
latora przez argument stały (8-bitowa liczba)
A <- A ∪ dana
kod: 0 1 0 0 0 1 0 0 44h
cykle: 1 bajty: 2 (kod instrukcji
44h + dana)
przykład: ORL A, #23 (dodanie logiczne
A i liczby 23)

– ORL adres, A

dodana logicznie zostaje zawartość akumu-
latora i komórki o podanym adresie
„adres”, wynik zostaje umieszczony w ko-
mórce pamięci o adresie „adres”
(adres) <- (adres) ∪ A
kod: 0 1 0 0 0 0 1 1 42h
cykle: 1 bajty: 2 (kod instrukcji
42h + adres)
przykład: ORL A, 20h (dodanie logiczne
A i zawartości komórki pod adresem 20h)

– ORL adres, #dana

dodana logicznie zostaje zawartość komó-
rki o adresie „adres” oraz argument stały
(8-bitowa liczba)
(adres) <- (adres) ∪ dana
kod: 0 1 0 0 0 0 1 1 43h
cykle: 2 bajty: 3 (kod instrukcji
43h + adres + dana)
przykład: ORL 12h, #99 (dodanie logiczne
zawartości komórki 12h i liczby 99)

"XRL"

- a) ang. „logical XOR” – zsumuj mod 2 (różnica symetryczna)
- b) wykonywana jest suma mod 2 XOR wskazanych w instrukcji dwóch argumentów. Wynik operacji jest wpisywany do argumentu pierwszego instrukcji
- c) znaczniki: nie zmieniają się
- d) rodzaje instrukcji:

– XRL A, Rn

zsumowana (mod 2) zostaje zawartość akumulatora i rejestru Rn, wynik w A
 $A \leftarrow A \oplus Rn$ gdzie Rn = R0...R7 (jeden z rejestrów roboczych)
 kod: 0 1 1 0 1 n2 n1 n0 gdzie n2...n0 – wskazują na R0...7 stąd: 68h-6Fh
 cykl: 1 bajty: 1
 przykład: XRL A, R7

– XRL A, adres

zsumowana (mod 2) logicznie zostaje zawartość akumulatora i komórki o podanym adresie „adres”, wynik zostaje umieszczony w A
 $A \leftarrow A \oplus (adres)$
 kod: 0 1 1 0 0 1 0 1 65h
 cykl: 1 bajty: 2 (kod instrukcji 65h + adres)
 przykład: XRL A, 19h (dodanie logiczne A i zawartości komórki pod adresem 19h)

– XRL A, @Ri

zsumowana (mod 2) logicznie zostaje zawartość akumulatora komórki wewn. RAM o adresie wskazywanym przez rejestr Ri. (R0 lub R1)
 $A \leftarrow A \oplus (Ri)$
 kod: 0 1 1 0 0 1 1 i gdzie i wskazuje na R0 (i=0) lub R1 (i=1) stąd: 66h, 67h
 cykl: 1 bajty: 1
 przykład: XRL A, @R0 (zsumowanie (mod 2) logiczne A i zawartości komórki o adresie w R0)

– XRL A, #dana

zsumowana (mod 2) logicznie zostaje zawartość akumulatora przez argument stały (8-bitowa liczba)
 $A \leftarrow A \oplus dana$
 kod: 0 1 1 0 0 1 0 0 64h
 cykl: 1 bajty: 2 (kod instrukcji 64h + dana)
 przykład: XRL A, #23 (zsumowanie (mod 2) logiczne A i liczby 23)

– XRL adres, A

zsumowana (mod 2) logicznie zostaje zawartość akumulatora i komórki o podanym adresie „adres”, wynik zostaje umieszczony w komórce pamięci o adresie „adres”
 $(adres) \leftarrow (adres) \oplus A$
 kod: 0 1 1 0 0 0 1 1 62h
 cykl: 1 bajty: 2 (kod instrukcji 62h + adres)
 przykład: XRL A, 20h (zsumowanie (mod 2) logiczne A i zawartości komórki pod adresem 20h)

– XRL adres, #dana

zsumowana (mod 2) logicznie zostaje zawartość komórki o adresie „adres” oraz argument stały (8-bitowa liczba)
 $(adres) \leftarrow (adres) \oplus dana$
 kod: 0 1 1 0 0 0 1 1 63h
 cykl: 2 bajty: 3 (kod instrukcji 63h + adres + dana)
 przykład: XRL 12h, #99 (dodanie logiczne zawartości komórki 12h i liczby 99)

"CLR A"

- a) ang. „clear accumulator” – zeruj akumulator
- b) do akumulatora zostaje wpisana wartość 0.
- c) znaczniki: bez zmian

d) A <- 0

kod: 1 1 1 0 0 1 0 0 E4h
 cykl: 1 bajty: 1
 Przykład: efekt wyzerowania akumulatora można uzyskać stosując instrukcję:
 MOV A, #0
 lecz w tym przypadku instrukcja ma długość 2 bajtów (a CLR A tylko 1), co w efekcie skraca długość kodu programu i oszczędza pamięć.

"CPL A"

- a) ang. „complement accumulator” – zaneguj akumulator
- b) wartość akumulatora zostaje zanegowana, wynik wpisany zostaje do akumulatora
- c) znaczniki: bez zmian
- d) $A \leftarrow \neg A$
 kod: 1 1 1 0 1 0 1 0 F4h
 cykl: 1 bajty: 1
 Przykład: aby np. zmienić znak liczby zapisanej w akumulatorze w kodzie U2 należy wykonać sekwencję instrukcji:
 CPL A ;negacja akumulatora
 INC A ;inkrementacja akumulatora

"RL A"

- a) ang. „rotate left” – przesunij w lewo
- b) zawartość akumulatora zostaje przesunięta w lewo o 1 pozycję (o 1 bit), to znaczy że:
 bit 1 przyjmuje wartość bitu 0
 bit 2 przyjmuje wartość bitu 1
 itd.....
 bit 7 przyjmuje wartość bitu 6
 a bit 0 przyjmuje wartość bitu 7
- c) znaczniki: bez zmian
- d) $A \leftarrow rotacja\ w\ lewo\ (A)$
 kod: 0 0 1 0 0 0 1 1 23h
 cykl: 1 bajty: 1
 Przykład: jeżeli w A jest liczba 43h (01000011 binarnie) to po wykonaniu instrukcji:
 RL A
 akumulator będzie zawierał liczbę: 86h (10001110 binarnie).

"RLC A"

- a) ang. „rotate left through carry” – przesunij cyklicznie w lewo ze znacznikiem C
- b) zawartość akumulatora zostaje przesunięta w lewo o 1 pozycję (o 1 bit) z uwzględnieniem znacznika C, to znaczy że: znacznik C przyjmuje wartość bitu 7 (akumulatora oczywiście)
 bit 1 przyjmuje wartość bitu 0
 bit 2 przyjmuje wartość bitu 1
 itd.....
 bit 7 przyjmuje wartość bitu 6
 a bit 0 przyjmuje wartość znacznika C
- c) znaczniki: C jest ustawiany zgodnie z wynikiem operacji
- d) $A \leftarrow rotacja\ w\ lewo\ (A)\ z\ C$
 kod: 0 0 1 1 0 0 1 1 33h
 cykl: 1 bajty: 1
 Przykład: jeżeli w A jest liczba 54h (01010100 binarnie) to wykonanie instrukcji:
 CLR C ;wyzeruje znacznik C
 RLC A ;przesunij w lewo z C
 ;akumulator

 spowoduje wymnożenie przez 2 liczby zapisanej w naturalnym kodzie binarnym w akumulatorze.

"RR A"

- a) ang. „rotate right” – przesunij w prawo
- b) zawartość akumulatora zostaje przesunięta w prawo o 1 pozycję (o 1 bit), to znaczy że:
 bit 0 przyjmuje wartość bitu 1

bit 1 przyjmuje wartość bitu 2
 itd.....

bit 6 przyjmuje wartość bitu 7

a bit 7 przyjmuje wartość bitu 0

- c) znaczniki: bez zmian
- d) $A \leftarrow rotacja\ w\ prawo\ (A)$
 kod: 0 0 0 0 0 1 1 03h
 cykl: 1 bajty: 1
 Przykład: jeżeli w A jest liczba 43h (01000011 binarnie) to po wykonaniu instrukcji
 RR A
 akumulator będzie zawierał liczbę: A1h (10100001 binarnie).

"RRC A"

- a) ang. „rotate right through carry” – przesunij cyklicznie w prawo ze znacznikiem C
- b) zawartość akumulatora zostaje przesunięta w prawo o 1 pozycję (o 1 bit) z uwzględnieniem znacznika C, to znaczy że: znacznik C przyjmuje wartość bitu 0 (akumulatora oczywiście)
 bit 0 przyjmuje wartość bitu 1
 bit 1 przyjmuje wartość bitu 2
 itd.....
 bit 6 przyjmuje wartość bitu 7
 a bit 7 przyjmuje wartość znacznika C
- c) znaczniki: C jest ustawiany zgodnie z wynikiem operacji
- d) $A \leftarrow rotacja\ w\ prawo\ (A)\ z\ C$
 kod: 0 0 0 1 0 0 1 1 13h
 cykl: 1 bajty: 1
 Przykład: jeżeli w A jest liczba 54h (01010100 binarnie) to wykonanie instrukcji:
 CLR C ;wyzeruje znacznik C
 RLC A ;przesunij w lewo z C akumulator

 spowoduje podzielenie przez 2 liczby zapisanej w naturalnym kodzie binarnym w akumulatorze.

"SWAP A"

- a) ang. „swap nibbles within accumulator” – wymień półbajty w akumulatorze
- b) w wyniku tej instrukcji wymieniona zostaje zawartość bitów 3...0 (mniejszy znaczący półbajt) i bitów 7...4 (bardziej znaczący półbajt) akumulatora. Operacja ta jest równoważna 4-krotnemu przesunięciu zawartości akumulatora.
- c) znaczniki: bez zmian
- d) $A_{3-0} \leftrightarrow A_{7-4}$
 kod: 1 1 0 0 0 1 0 0 C4h
 cykl: 1 bajty: 1
 Przykład: sekwencja podana poniżej powoduje zamianę półbajtów akumulatora
 MOV A, #52h ;wpisanie do akumulatora liczby 52h
 SWAP A ;wykonanie polecenia zamiany
 ;w akumulatorze znajduje się teraz liczba 25h

Uff! Na razie to tyle w następnym odcinku dokończenie listy instrukcji, a więc pozostałe komendy dotyczące:

- operacji przemieszczania danych
- operacji na bitach (znacznikach)
- skoki i pozostałe

oraz krótki opis asemblera ASM51 przeznaczony szczególnie dla komputerowców.

Ślawomir Surowiński

Lekcja 2

W dzisiejszej lekcji sprawdzimy działanie niektórych spośród omówionych wcześniej instrukcji arytmetycznych i logicznych procesora na podstawie przykładowego programu. Działanie programu jest bardzo proste, otóż:

a) najpierw program prosi o wprowadzenie dwóch liczb 8-bitowych w postaci heksadecymalnej,

czyli z zakresu 0...FFh (0...255 dziesiętnie). Pierwsza liczba wyświetlana jest na wyświetlaczach DL1 i DL2, druga na DL4 i DL5

b) następnie wykonywana jest wybrana przez Ciebie operacja arytmetyczna lub logiczna (o tym jak ją wybrać – za chwilę)

c) w efekcie na wyświetlaczach DL7 i DL8 wypisywany jest wynik operacji, który mo-

żesz sprawdzić ręcznie (na papierze) lub korzystając z kalkulatora wyposażonego w konwerter liczb zapisanych dziesiętnie i szesnastkowo (np. taki z MS-Windows).

Program w postaci listingu – czyli w zapisie źródłowym z dodatkowymi informacjami istotnymi szczególnie dla tych którzy nie mają komputera jest następujący:

```
;Program do lekcji nr 2
;testowanie komend: ADD, SUBB, ANL, ORL, XRL, SWAP
;z wykorzystaniem instrukcji BIOS'a

;*****
;
8000          org      8000h          ;poczek zewn. pamieci programu
;*****

8000 120274    znowu:    lcall    CLS          ;wyczyszczenie wyswietlacza
8003 75F001    mov      B,#1          ;pozycja 1 na displeju
8006 757840    mov      DL1,#_minus   ;znak "—" na pozycji wprowadzenia
8009 757940    mov      DL2,#_minus   ;pierwszego skladnika
800C 1203A7    lcall    GETACC         ;pobranie skladnika 1 dodawania
800F 128036    lcall    wait1         ;odczekaj sekunde
8012 C0E0      push    Acc            ;i przechowanie go na stosie
8014 75F004    mov      B,#4          ;pozycja 4 na displeju
8017 757B40    mov      DL4,#_minus   ;znak "—" na pozycji wprowadzenia
801A 757C40    mov      DL5,#_minus   ;drugiego skladnika
801D 1203A7    lcall    GETACC         ;pobranie skladnika 2 dodawania
8020 128036    lcall    wait1         ;odczekaj sekunde
8023 D0F0      pop     B              ;sciagniecie skladnika 1 ze stosu do rej.B
8025 C3        clr     C              ;potrzebne do testowania instrukcji SUBB
8026 25F0      add     A,B            ;komenda dodania skladnikow - tu wstaw inne komendy
8028 75F007    mov      B,#7          ;pozycja 7 na displeju
802B 12024E    lcall    A2HEX         ;wypisanie wyniku dodawania
802E 128036    lcall    wait1         ;odczekaj sekunde
8031 128036    lcall    wait1         ;odczekaj sekunde
8034 80CA      sjmp    znowu          ;i nastepne skladniki

;*****
;
8036 C0E0      wait1:  push    Acc          ;przechowanie A na stosie
8038 74FF      mov     A,#255
803A 120295    lcall    DELAY           ;odczekanie 0,5 sek
803D 74FF      mov     A,#255
803F 120295    lcall    DELAY           ;odczekanie 0,5 sek (w sumie 1 sek.)
8042 D0E0      pop     Acc              ;odtworzenie A (ze stosu)
8044 22        ret                     ;powrot do programu glownego
;*****

8045          END
```

Szczegółowy opis listingu nie jest tematem niniejszej lekcji (a przyszłego odcinka szkoły mikroprocesorowej), toteż przedstawię tylko istotne informacje potrzebne do wykonania zadania z naszej dzisiejszej lekcji. Informacje podzielę na te istotne dla komputerowców oraz dla „ręczniaków” (o ile mogą posłużyć się takim skrótem), tak więc, patrzymy na listing powyżej i wyjaśniamy sobie:

a) w pierwszej kolumnie podany jest adres początkowy danej linii programu z zawartą w niej instrukcją. U nas adres początkowy to 8000h – początek pamięci SRAM w komputerku. Zauważmy że cały program zajmuje 45 bajtów, bo ostatnim adresem jest 8045h – ostatnia linia listingu.

b) w każdej zawierającej instrukcję linii tuż za adresem znajduje się ciąg bajtów będący odpowiednikiem maszyno-

wym instrukcji zapisanej w dalszej części linii w sposób jawny. Dzięki temu „niekomputerowcy” będą mogli po prostu wklepać te dane „ciurkiem” od adresu 8000h bez mozolnego tłumaczenia z postaci źródłowej znajdującej się w trzeciej kolumnie listingu). Warto jednak przy tym chociaż chwilę zastanowić się i przetłumaczyć już teraz (korzystając z tabeli we wkładce) znane i nieznane instrukcje w kolej-

nich liniach a następnie porównać je z danymi z kolumny drugiej.

c) w listingu występują odwołania do procedur umieszczonych w programie monitora są to:

GETACC, A2HEX i DELAY. Nie wdając się w szczegóły (na razie) wyjaśniam że nazwom tym przypisane są adresy w przestrzeni programu monitora (EPROM) od których zaczynają się kody tych procedur. Ich działanie jest następujące: "GETACC" : procedura pobrania, z klawiatury, 8-bitowej liczby zapisanej w postaci szesnastkowej i umieszczenie jej w akumulatorze z jednoczesnym wyświetleniem wpisywanej przez użytkownika wartości na wyświetlaczu. Pozycja na której wypisywana jest wartość na displeju powinna być określona przed jej wywołaniem w rejestrze B. W naszym przykładzie dzięki tej procedurze możesz wprowadzić składniki testowanego działania.

"A2HEX" : procedura wyświetlenia na displeju w postaci szesnastkowej (na 2 wyświetlaczach) aktualnej zawartości akumulatora. Podobnie jak poprzednio pozycja od której ma być wypisana liczba musi być określona w rejestrze B. W naszym przykładzie dzięki tej procedurze wyświetlany jest wynik operacji na wyświetlaczach DL7 i DL8.

"DELAY" : procedura opóźnienia, czas opóźnienia jest podawany w akumulatorze przed wywołaniem procedury a mnożnik wynosi około 1,95 ms. Jeżeli zatem wpiszesz do akumulatora wartość 255 to po wywołanie procedury DELAY będzie trwało ok. $255 \times 1,95 \text{ ms} = 497 \text{ ms}$ czyli około 0,5 sekundy. Zastosowanie tej procedury w naszym przykładzie ma umożliwić Ci obserwację wykonywania programu „krok po kroku”.

d) dowolny tekst znajdujący się za znakiem średnika „;” jest traktowany jako komentarz i nie jest brany pod uwagę podczas kompilacji programu w przypadku korzystania z kompilatora na komputer PC. Uwagi zawarte w komentarzu są bardzo przydatne podczas analizy programu.

e) w przykładowym listingu większość instrukcji jest Ci jeszcze nie znana, są one jednak niezbędne do wykonania tej lekcji, toteż proszę traktuj je jako domysłne, więcej informacji na ich temat w kolejnym numerze EdW.

Dla „niekomputerowców”:

Korzystając z funkcji monitora „Edit” – należy wklepać kod programu, korzystając z listingu powyżej, od adresu 8000h. Dla ułatwienia podaję że pierwsze bajty kodu to:

12, 02, 74, 75, F0, 01, 75, 78, 40, 75, 79, 40, 12, 03, A7, 12, 80, 36, C0, E0 itd....

Wytrwałym proponuję analizę i przetłumaczenie (tabela instrukcji we wkładce) kilku pierwszych lub całego listingu programu a następnie porównanie efektów swojej pracy z kodami podanymi w naszym przykładzie.

f) w naszym listingu w linii o adresie 8026h znajduje się właściwa instrukcja testująca działanie danej funkcji arytmetyczno-logicznej (zaciemniona linia). W przykładzie naszym znajduje się instrukcja ADD – dodawania.

W przypadku chęci zastosowanie innej funkcji należy w miejsce kodu „25 F0” (pod adresem 801A) wpisać odpowiednie dla poszczególnych instrukcji, ciągi bajtów:

| | |
|------------------|-------|
| dla SUBB wpisać: | 95 F0 |
| dla ANL wpisać: | 55 F0 |
| dla ORL wpisać: | 45 F0 |
| dla XRL wpisać: | 65 F0 |

korzystając z funkcji „Edit” monitora.

Uwaga, w przypadku testowania instrukcji SWAP A należy wpisać liczby: C4 00.

Zauważmy wszakże że poprzednie instrukcje były dwubajtowe, ta ostatnia zaś jest 1-bajtowa. Dlatego na pozycji drugiego bajtu wpisałem 00 co jest kodem maszynowym instrukcji „NOP” – „nie rób nic”. Instrukcję NOP poznasz dokładnie w kolejnym numerze EdW. Na razie powiem Ci tylko że podczas wykonywania instrukcji NOP procesor nie robi nic – czyli de facto leniuchuje przez jeden cykl maszynowy (12 cykli zegara). Zastosowanie tej instrukcji przy modyfikacji kodu z poziomu monitora (funkcja Edit) jest uzasadnione, bo wiem jest ona niejako „wypełniaczem” brakującego bajtu kodu o adresie: 8027h. Przy wprowadzaniu danych podczas testu instrukcji SWAP pierwszy składnik nie jest brany pod uwagę (bo instrukcja SWAP jest 1-argumentowa), toteż można wpisać dowolną wartość najlepiej 00.

Po modyfikacjach należy opuścić funkcję „Edit” i uruchomić ponownie program – komenda monitora „Jump”, a następnie sprawdzić działanie nowo wprowadzonej instrukcji.

Dla komputerowców:

Uwaga: przy przeglądaniu i modyfikacjach naszego przykładu korzystaj z DOSowego Nortona Commandera! Zanim zaczniesz zabawę przeczytaj uważnie plik informacyjny ASM51.DOC.

Na dyskiecie z kompilatorem ASM51 znajduje się zbiór źródłowy z naszym przykładem pod nazwą „LEKCJA2.S03”. Powinieneś wykonać następujące czynności:

- skompilować przykład do postaci maszynowej, skorzystaj z programu wsadowego „DO.BAT”, wydaj komendę:
`> DO LEKCJA2 {Enter}`
- załaduj program do komputerka (komenda „Load” monitora)

- uruchom program (komenda „Jump”) najpierw z instrukcją arytmetyczną ADD (domyślnie znajduje się w pliku LEKCJA2.S03)

- wykonaj kilka działań na przykładowych liczbach

- zmodyfikuj plik źródłowy (klawisz F4 w Nortonie) czyli linię z instrukcją ADD zamień na inne instrukcje podane wcześniej w ćwiczeniu: SUBB, ANL, ORL, XRL, wpisując je w miejsce ADD (uwaga: wielkość liter nie ma znaczenia)

- skompiluj ponownie program i załaduj do komputerka

- uruchom ponownie program i wykonaj kilka działań sprawdzając na kartce papieru lub kalkulatorze poprawność działania poszczególnych instrukcji.

Przy okazji zajrzyj do powstałych w wyniku kompilacji zbiorów:

- listingu : LEKCJA2.LST

- zbioru : LEKCJA2.HEX zapisanego w formacie Intel HEX. Więcej na temat tego formatu danych możesz dowiedzieć się z artykułu w naszym bratnim piśmie „Elektronika Praktyczna” nr 10/97 na stronie 75. Na łamach naszego kursu wrócimy przy innej okazji do tego tematu.

Postaraj się zapoznać ze zbiorem typu listing. Przekształć „ręcznie” dowolne linie programu (korzystając z tabeli we wkładce w tym numerze EdW) i porównaj otrzymane kody poszczególnych linii z tymi w zbiorze LEKCJA2.LST.

Jako przykłady proponuję wykonać następujące operacje:

a) test funkcji ADD:

argumenty: 12h, 67h wynik: 79h

argumenty: FEh, 02h wynik: 00h

b) test funkcji SUBB: (uwaga: tu 1-szy argument jest odejmowany od 2-go !)

argumenty: 12h, 67h wynik: 55h

argumenty: F0h, 05h wynik: 15h

c) test funkcji ANL:

argumenty: 1Fh, EEh wynik: 0Eh

argumenty: F0h, 0Fh wynik: 00h

d) test funkcji ORL:

argumenty: 7Eh, 80h wynik: FEh

argumenty: 70h, 09h wynik: 79h

e) test funkcji XRL:

argumenty: 25h, 6Bh wynik: 4Eh

argumenty: 55h, AAh wynik: FFh

f) test funkcji SWAP:

argumenty: pierwszy nie istotny, 78h

wynik: 87h

argumenty: pierwszy nie istotny, 39h

wynik: 93h

Zauważ że wszystkie instrukcje działają na liczbach 8-bitowych, toteż w przypadku przekroczenia zakresu tych liczb informacja o wyniku jest częściowo tracona.

Zycząc wesołej zabawy i dużo wytrwałości !

Sławomir Surowiński



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 monotony, 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” – przesunąć, przenieść).

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
- cykl: 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
- cykl: 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
- cykl: 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
- cykl: 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
- cykl: 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
- cykl: 1 bajty: 2 (kod instrukcji + adres)
- przykład:

Też to potrafisz

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 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 nią ;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 MOV

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 pamięci 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 niezmiennie, 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 przeznaczonej pamięci umieścić 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 EPROM (ROM) i odczytać ją na dowolnym programatorze (bez stosowania instrukcji MOV).

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

stępną 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 MOVX 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 zewnętrzną pamięć danych można zaadresować w dwójki 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 pełną kontrolę sposobu i kierunku ustawienia jego końcówek – a więc jest metodą na maksymalne wykorzystanie cechy „jednokładowości” procesora.

I tak dwa wspomniane dwa tryby adresowania zewnętrznej RAM mają 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 należy wykonać instrukcję:

```
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 0 0 0 0 0 F0h
- cykle: 2 bajty: 1
- przykład:
aby zapisać daną w obszarze zewnętrznej pamięci danych pod adresem np. 8000h należy wykonać instrukcję:

```
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

- 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 zatrasku 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 instrukcję:

```
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 por-

tu 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 0 0 0 1 i gdzie i = 0, 1

- cykle: 2 bajty: 1

- przykład:
weźmy sytuację 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 wewn. RAM wskazywanym przez SP) zostaje zapisana zawartość komórki z wewn. 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 „zdejmienia ze stosu”.

Też to potrafisz

(adres) \leftarrow (SP), SP \leftarrow 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. Dodatkowo 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 \leftrightarrow 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 \leftrightarrow (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 \leftrightarrow (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 R0 (i=0) lub R1 (i=1).

A₃₋₀ \leftrightarrow (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 dwójako:

- 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 \leftarrow 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 \leftarrow 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) \leftarrow 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) \leftarrow 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 \leftarrow /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) \leftarrow /(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 powinienś 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 \leftarrow C \wedge (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)

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
 $C \leftarrow C \cap /(\text{bit})$
- c) kod: 1 0 1 1 0 0 0 0 B0h
- d) cykl: 2 bajty: 2 (kod instrukcji + adres bitu)
- e) przykład:
 ANL C, /D7h ;równoważne wyzerowaniu znacznika C ;bo zmnożono 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
 $C \leftarrow C \cup (\text{bit})$
- c) kod: 0 1 1 1 0 0 1 0 72h
- d) cykl: 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
 $C \leftarrow C \cup /(\text{bit})$

- c) kod: 1 0 1 0 0 0 0 0 A0h
- d) cykl: 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”, przeniesz wartość 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
 $C \leftarrow (\text{bit})$
- c) kod: 1 0 1 0 0 0 1 0 A2h
- d) cykl: 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”, przeniesz wartość 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”
 $(\text{bit}) \leftarrow C$
- c) kod: 1 0 0 1 0 0 1 0 92h
- d) cykl: 2 bajty: 2 (kod instrukcji + adres bitu)
- e) przykład: – pytanie?...
 jak przeniesć zawartość jakiegoś bitu (o podanym adresie bezpośrednim do innego o innym adresie bezpośrednim – czyli wykonać operację :
 $(\text{bit}2) \leftarrow (\text{bit}1)$
 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 bezpośredniego.

Załóżmy że bit1 zdefiniowaliśmy jako:
 bit1 EQU 20h ;EQU to deklaracja równoważności – przypisanie ;nazwie umownej (wyrazowi) konkretnego adresu

a bit 2 jako
 bit2 EQU 30h
 Chcąc teraz przenieść zawartość bitu 1 do bitu 2 częścią początkującą 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 wykorzystanie 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 przeciwwieżenia 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 płycie komputera – 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.

Popatrzymy zatem i przeanalizujemy 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 jedynek 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 komputer zawiesiłby się.

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

Też to potrafisz

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

Uwaga, przed rozpoczęciem wpisywania programu upewnij się czy zworka JP3 znajduje się w pozycji „8000h”.

Dla komputerowców:

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

Należy go skompilować jak poprzednie lekcje wydając komendę

DO LEKCJA3 lub użyć bezpośredniego kompilatora:

> PASM51 LEKCJA3.S03 /H /L
co spowoduje utworzenie zbioru wynikowego w postaci Intel–HEX, gotowego do przesłania do komputerka poprzez łącze szeregowo.

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 miejscu 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 mogą zmodyfikować tę linię w programie LEKCJA3.S03

np. mov A, #99h

.....

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

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

Wesołej zabawy !

Ślawomir Surowiński

CPU 8052.DEF

```

;*****
; Lekcja 3: Testowanie zewnętrznej pamięci danych U4
;*****
include const.inc
include bios.inc                                ;deklaracje dla kompilatora

8000                                     org      8000h                                ;kod programu od adresu 8000h

8000 120274                             lcall   CLS                                ;wyczyszczenie wyświetlacza
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 zmianę adresu
800B 75F001                             mov     B,#1                                ;od pozycji 1 wyświetlacza
800E 12025F                             lcall   DPTR4HEX                         ;wypisz aktualnie testowany adres RAM
8011                                     test55:

8011 7455                               mov     A,#55h                            ;załadowanie 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                            ;załadowanie 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                            ;zwiększenie wskaźnika 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 następna komorka

8028                                     koniec:

8028 75D79                             mov     DL6,#_E                            ;na koniec milutki napis „End” – koniec
802B 75E54                             mov     DL7,#_n                            ;informujący że koniec testu
802E 75F5E                             mov     DL8,#_D                            ;i skok na koniec programu
8031 8009                             sjmp    stop

;-----
;ta czesc programu zadziala gdy bedzie blad jakiegos komorki RAM

8033 75D79                             blad: mov     DL6,#_E
8036 75E50                             mov     DL7,#_r
8039 75F50                             mov     DL8,#_r
803C 75F1E0                           stop: mov     blinks,#E0h                            ;jest blad pamieci RAM – napis „Err”
                                                    ;aby napis „End” lub „Err” bedzie mrygal

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

8041                                     END
```


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 informujemy 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
- 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 nastąpi 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 pomocą 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. Programiści w takim przypad-

Też to potrafisz

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
              ;skocz do etykiety „zeruj”
Podany przykład, z praktycznego punktu wi-
dzenia 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 zero-
wanie 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 wywołaj 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<sub>7-0</sub>, SP ← SP + 1, (SP) ← PC<sub>15-8</sub>, PC<sub>10-0</sub> ← 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 sek-
;wencji 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 tekstu
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 wyslij 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) ← PC₇₋₀, SP ← SP + 1, (SP) ← PC₁₅₋₈, PC₁₅₋₀ ← 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 komputera edukacyjnego przy pomocy instrukcji A2HEX standardowo umieszczonej w kodzie monitora systemu (w EP-ROM 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
          ;pod adresem
          ;024Eh znajduje
          ;się gotowy
          ;podprogram do wypisywania
          ;akumulatora w
          ;postaci 2 cyfr HEX
          ;W efekcie wykonania przykładu na wy-
          ;świetlaczach zostanie wypisana liczba 12h.
          ;W postaci „12”.
```

RET

- ang. „return from subroutine”, powrót z podprogramu
 - adres powrotu z podprogramu (zapamiętana wcześniej podczas instrukcji ACALL lub LCALL wartość licznika PC) jest wypisywany do licznika rozkazów PC. Wskaźnik stosu zostaje mniejszony o 2. W efekcie następuje powrót z podprogramu. Instrukcją tą musi kończyć się każdy podprogram (z wyjątkiem podprogramów obsługi przerwań, wywoływanych automatycznie przez procesor).
- PC₁₅₋₈ ← (SP), SP ← SP – 1
 PC₇₋₀ ← (SP), SP ← SP – 1
- kod: 0 0 1 0 0 0 1 0 22h
 - cykl: 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”, powrót 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.
- PC₁₅₋₈ ← (SP), SP ← SP – 1
 PC₇₋₀ ← (SP), SP ← SP – 1
- kod: 0 0 1 1 0 0 1 0 32h
 - cykl: 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. Różnica między nimi polega na „odległości” (o ile bajtów programu dalej można skoczyć) pomiędzy instrukcją skoku a miejscem w którym następuje skok. I tak mamy do czynienia ze skokiem:

- krótkim (instrukcja SJMP – „short jump”), gdzie skoczyć możemy w obrębie 256 bajtów (w górę o 127 lub w dół o 128) od miejsca wy-

wolania instrukcji SJMP. W tym przypadku argumentem instrukcji jest przesunięcie (tak jak wspomniany wcześniej parametr „rel” przy okazji omawiania instrukcji skoków warunkowych operujących na bitach), wyrażone poprzez liczbę 8-bitową zapisaną w kodzie U2. na stronie (instrukcja AJMP – „absolute jump on page”), gdzie skok może nastąpić w obszarze 2 kilobajtowej strony. W tym przypadku argumentem instrukcji 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 nastąpić 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 wystarczyłoby 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 się objętością zajmowaną w kodzie programu. I tak skok krótki SJMP oraz na stronie AJMP zajmują 2 bajty kodu, natomiast skok długi LJMP zajmuje 3 bajty, ktoś powie że to mała różnica, otóż 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 szczegółów.

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.
- PC₁₀₋₀ ← adr11
- kod: a10 a9 a8 0 0 0 0 1 1 ,gdzie: a10, a9, a8 zależą od numeru strony

stad: 01h, 21h, 41h, 61h, 81h, A1h, C1h, E1h

- cykl: 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 omiinie instrukcję 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
 - cykl: 2 bajty: 2 (kod instrukcji 80h + przesunięcie „rel”)
 - przykład:
- ```
MOV A,B ;dowolne instrukcje
CLR B ;programu
.....
stop: SJMP stop ;i przykład pętli nieskończonej, którą
 ;powinien dla
 ;bezpieczeństwa kończyć
 ;się każdy program
 ;deklaracja końca
 ;programu (to nie jest
 ;instrukcja!)
```
- END

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

Dokończenie listy instrukcji w następnym numerze.

Sławomir Surowiński

# Lekcja 4

W dzisiejszej lekcji poćwiczmy 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 EP-

ROM 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 zastanawialiście się drogi Czytelniku, jak to się dzieje, że po naciśnięciu przez Ciebie odpowiedniego klawisza, układ reaguje



## Też to potrafisz

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 ograniczoną objętość artykułu, w dzisiejszej lekcji wykorzystamy tylko niektóre procedury (podprogramy).

Dla przykładu skorzystamy 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 wyświetlenia                          |
| 0003        | pozycja | equ         | 3    | ;pozycja na której ma być<br>;wyświetlona liczba |
| 8000        | org     | 8000h       |      |                                                  |
| 8000 12074  | lcall   | CLS         |      | ;wyczyszczenie wyświetlacza                      |
| 8003 7445   | mov     | A, #liczba  |      | ;załadowanie liczby                              |
| 8005 75F003 | mov     | B, #pozycja |      | ;i pozycji na displayu                           |
| 8008 1204E  | lcall   | A2HEX       |      | ;wyświetlenie                                    |
| 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” komputera.

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 profilaktycznie 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, bo w końcu nie korzystają z mądrego kompilatora,

czyli wspomniana liczba 45h – prawda że się zgadza!

L7: musimy także powiedzieć procedurze A2HEX, żeby wyświetliła liczbę od pozycji 3 na displayu, 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 **024E**, 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 kółko 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 programu spoza adresu 800Dh, a tam co jest?, a no same „śmieci” Kochani – sprawdźcie to funkcją EDIT monitora.

Nie muszę 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 komputer. 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 mogli naprawdę „poszaleć”. Nabywcy dyskietek AVT2250/D mogą się z nimi już teraz zapoznać czytając zbiory BIOS.INC i CONST.INC.

Wesołej zabawy!

Sławomir Surowiński



**JMP @A+DPTR**

- ang. „Jump Indirect relative to DPTR” – skoczek 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. Przesunięcie 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  
 – cykl: 2 bajty: 1  
 – przykład: realizacja skoków w miejscu w zmiennej „pozycja” (umieszczonej w pamięci RAM procesora)

MOV A, pozycja ;załadowanie adresu  
 MOV B, #2 ;2 bajty w tabeli skoków sa 2-bajtowe (AJMP)

MUL A, B ;obliczenie faktycznego offsetu do tabeli skoków  
 MOV DPTR, #tablica\_skokow ;załadowanie adresu tabeli skoków do DPTR

JMP @A+DPTR;wykonanie skoku  
 tablica\_skokow: ;tu zaczyna sie tabela skoków

AJMP etyk01 ;tu nastąpi skok gdy „pozycja” = 0

AJMP etyk02 ;tu nastąpi skok gdy „pozycja” = 1

AJMP etyk03 ;tu nastąpi skok gdy „pozycja” = 2

AJMP etyk04 ;tu nastąpi skok gdy „pozycja” = 3

..... ;pozostałe instrukcje programu

etyk01: ;właściwe miejsce skoku według pozycji = 0  
 ..... ;tu można umieścić

etyk02:

instrukcje  
 ;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 pętach programowych, gdzie konieczne jest wykonanie n-razy określonej operacji.

**JZ rel**

- ang. „Jump if Accumulator is Zero”, skoczek 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 wcześniej 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  
 – cykl: 2 bajty: 2 (kod instrukcji 60h + przesunięcie „rel”)

– przykład:  
 MOV A, B ;załadowanie rej. B do Acc celem sprawdzenia czy = 0

JZ jest\_zero ;jeżeli tak to skok do etykiety „jest\_zero”  
 nie\_zero: ;jeżeli nie to wykonuj po pozostałe instrukcje

.....

.....

jest\_zero:

;tu nastąpi skok jeżeli rej.B był równy zero  
 ;i zostaną wykonane te instrukcje

.....

**JNZ rel**

- ang. „Jump if Accumulator is Not Zero”, skoczek 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 wcześniej 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  
 – cykl: 2 bajty: 2 (kod instrukcji 70h + przesunięcie „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 pozostałe instrukcje

.....

.....

nie\_zero:

;tu nastąpi skok jeżeli rej.B był różny od zera  
 ;i zostaną wykonane te instrukcje

.....

A oto wspomniana wcześniej grupa instrukcji używana głównie w pętach programowych lub przy wielokrotnionym sprawdzaniu warunków zgodności 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 skoczek jeżeli argumenty porównania nie są sobie równe
- W instrukcji tej porównywane są dwa argumenty: arg1 i arg2. Jeżeli nie są one równe (ich wartości nie są równe), to do zawartości licznika rozkazów jest dodawane przesunięcie „rel” na zasadach zgodnych z omówionymi wcześniej. W efekcie zostaje wykonany skok w programie. Tu uwaga, skok następuje względem instrukcji występującej po instrukcji CJNE. Dodatkowo jest zmieniany znacznik przeniesienia C. I tak, jeżeli w wyniku porównania okaże się że argument 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 zależności od typu argumentów instrukcji możliwe są cztery przypadki, oto one.

**CJNE A, adres, rel**

- porównywany jest akumulator oraz komórka wew. RAM o adresie podanym bezpośrednio jako argument „adres”
- PC ← PC + 3, jeśli 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

– cykl: 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. Następnie sprawdzany jest warunek zgodności akumulatora z rejestrem B, jeżeli nie są sobie równe, to akumu-

lator 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 następuje 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

– cykl: 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

## Też to potrafisz

w zależności od rodzaju klawisza, należy przechowywany kod klawisza porównać z konkretną liczbą.

czekaj:

```
MOV A, klawisz ;pobranie kody
 wciśniętego
 klawisza
CJNE A, #65, sprB ;czy wciśnięto
 klawisz „A” (65
 - kod „A”)
 ;tak to wykonuj
 te instrukcje
.....
```

sprB:

```
CJNE A, #66, sprC ;nie to czy
 wciśnięto
 klawisz „B”
 ;tak to wykonuj
 te instrukcje
.....
```

sprC:

```
CJNE A, #67, sprD ;nie to czy
 wciśnięto
 klawisz „C”
 ;tak to wykonuj
 te instrukcje
.....
```

sprD:

```
CJNE A, #68, czekaj ;nie to czekaj
 na kolejne
 naciśnięcie
 klawisza
 ;tak to wykonuj
 te instrukcje
.....
```

### CJNE Rn, #dana, rel

– rejestr Rn (R0...R7) zostaje porównany z argumentem bezpośrednim, jeżeli nie są zgodne zostaje wykonany skok

PC ← PC + 3, jeśli Rn <> dana, to PC  
← PC + rel gdzie n = 0...7  
dodatkowo: C ← 0 gdy Rn ≥ dana, lub  
C ← 1 gdy Rn < dana

– kod: 1 0 1 1 0 n2 n1 n0 gdzie n2 n1 n0  
określają jeden  
z rejestrów  
R0...R7

– cykl: 2 stąd kody: B8h...BFh  
bajty: 3 (kod instrukcji  
+ dana + przesuniecie „rel”)

– przykład:  
MOV R4, P1

odczytanie  
stanów z portu  
P1

CJNE R4, #100, nie\_100 ;porównanie  
ich z liczbą 100  
SETB P3.0 ;równe to  
ustaw pin 0  
portu P3

.....

nie\_100: CLR P3.0 ;nie równe to  
zeruj pin 0  
portu P3

.....

.....

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

### CJNE @Ri, #dana, rel

– porównywana jest zawartość komórki w wew. RAM której adres znajduje się w rejestrze Ri (R0 gdy i=0, lub R1 gdy i=1) z argumentem bezpośrednim. Jeżeli się różnią to następuje skok.

PC ← PC + 3, jeśli (Ri) <> dana, to PC  
← PC + rel gdzie i = 0, 1  
dodatkowo: C ← 0 gdy (Ri) ≥ dana, lub  
C ← 1 gdy (Ri) < dana

– kod: 1 0 1 1 0 1 1 i gdzie i=0,  
1 stąd kody:  
B6h, B7h

– cykl: 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 się 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 polecenie 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 zależności od typu argumentu rozróżnia się dwa typy instrukcji, oto one.

#### **DJNZ Rn, rel**

- zmniejszona zostaje zawartość podanego rejestru Rn (R0...R7) o jeden, a następnie jeżeli nie jest równa zero, to następuje skok.  
 $PC \leftarrow PC + 2$ ,  $Rn \leftarrow Rn - 1$ , gdzie  $n = 0...7$   
jeżeli  $Rn < 0$ , to  $PC \leftarrow PC + rel$
- kod: 1 1 0 1 1 n2 n1 n0 gdzie n2 n1 n0 określają jeden z rejestrów R0...R7 stąd kody: D8h...DFh
- cykl: 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 pamięci wew. RAM o podanym bezpośrednim adresie o jeden, a następnie jeżeli nie jest równa zero, to następuje skok.  
 $PC \leftarrow PC + 2$ ,  $(adres) \leftarrow (adres) - 1$ , jeżeli  $(adres) < 0$ , to  $PC \leftarrow PC + rel$
- kod: 1 1 0 1 0 1 0 1 D5h
- cykl: 2 bajty: 3 (kod instrukcji + adres + przesunięcie „rel”)
- przykład: sekwencja instrukcji przedstawiona poniżej daje taki sam efekt jak w poprzednim 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

## Też to potrafisz

- jest to instrukcja, w wyniku której nie zmienia się stan procesora, z wyjątkiem licznika rozkazów którym po pobraniu tej instrukcji jest zwiększany o jeden.
- kod: 0 0 0 0 0 0 0 0 00h
- cykle: 1 bajty: 1
- przykład: sekwencja instrukcji  
MOV A, B  
NOP  
MUL A, B  
będzie zajmie procesorowi 1 cykl maszynowy więcej niż sekwencja  
MOV A, B  
MUL A, B  
ale efekt działania będzie taki sam w obu przypadkach.

### LEKCJA 4

W dzisiejszej lekcji poćwiczmy sobie użycie instrukcji wywołań podprogramów do realizacji prostych układów licznikowych. 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)

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 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 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 na byli 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 ograniczoną 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.

#### Zadanie 1

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

Żałujemy 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 wpisać dowolną wartość np.
45h
MOV B, #3 ; na DL3
i DL4
LCALL A2HEX
```

Sławomir Surowiński





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

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

#### JMP A+DPTR

- ang. "Jump Indirect relative to DPTR" – skocz pośrednio względem rejestru DPTR
- w wyniku tej instrukcji następuje skok pod adres będący sumą aktualnej wartości rejestru DPTR (liczba 16-bitowa) i wartości akumulatora (liczba 8-bitowa). Można powiedzieć że skok następuje pod adres w pamięci programu umieszczony w DPTR z przesunięciem podanym w akumulatorze. Przesunięcie to traktowane jest jako liczba bez znaku, czyli z zakresu <0...255>

PC ← A + DPTR

– kod: 0 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 bajtowe (AJMP)
 ;obliczenie faktycznego offsetu
 ;do tabeli skoków
```

```
MUL A, B ;obliczenie faktycznego offsetu
 ;do tabeli skoków
```

```
MOV DPTR, #tablica_skokow ;załadowanie adresu
 ;tabeli skoków do DPTR
```

```
JMP A+DPTR ;wykonanie skoku
```

tablica\_skoków:

```
AJMP etyk01 ;tu zaczyna sie tabela skoków
```

```
AJMP etyk02 ;tu nastąpi skok gdy "pozycja" = 0
```

```
AJMP etyk03 ;tu nastąpi skok gdy "pozycja" = 1
```

```
AJMP etyk04 ;tu nastąpi skok gdy "pozycja" = 2
```

```
..... ;tu nastąpi skok gdy "pozycja" = 3
```

```
..... ;pozostałe instrukcje programu
```

```
etyk01: ;właściwe miejsce skoku według pozycji = 0
```

```
..... ;tu można 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 (rejestr) 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 operacji.

#### 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 wcześniej 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      60h

– cykle: 2      bajty: 2 (kod instrukcji 60h + przesunięcie "rel")

- przykład:

```
MOV A, B ;załadowanie rej. B do Acc celem
```

```
 ;sprawdzenia czy = 0
```

```
JZ jest_zero ;jeżeli tak to skok do etykiety "
```

```
 jest_zero"
```

```
nie_zero: ;jeżeli nie to wykonuj pozostałe instrukcje
```

```
.....
```

```
.....
```

```
jest_zero: ;tu nastąpi skok jeżeli rej.B był równy zero
```

```
.....
```

```
.....
```

```
;i zostaną wykonane te instrukcje
```

## Też to potrafisz

### 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 wcześniej 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 + przesunięcie "rel")

przykład:

```
MOV A, B ;załadowanie rej. B do Acc celem
 sprawdzenia czy = 0
JNZ nie_zero ;jeżeli nie to skok do etykiety "nie_zero"
jest_zero: ;jeżeli tak to wykonuj pozostałe instrukcje
```

.....

```
nie_zero: ;tu nastąpi skok jeżeli rej.B był różny od
 zera
```

```
..... ;i zostaną wykonane te instrukcje
.....
```

A oto wspomniana wcześniej grupa instrukcji używana głównie w pętlach programowych lub przy zwielokrotnionym sprawdzaniu warunków zgodności 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 są dwa argumenty: arg1 i arg2. Jeżeli nie są one równe (ich wartości nie są równe), to do zawartości licznika rozkazów jest dodawane przesunięcie "rel" na zasadach zgodnych z omówionymi wcześniej. W efekcie zostaje wykonany skok w programie. Tu uwaga, skok następuje względem instrukcji występującej po instrukcji CJNE. Dodatkowo jest zmieniany znacznik przeniesienia C. I tak, jeżeli w wyniku porównania okaże się że argument 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 zależności od typu argumentów instrukcji możliwe są cztery przypadki, oto one.

### CJNE A, adres, rel

- porównywany jest akumulator oraz komórka wew. RAM o adresie podanym bezpośrednio jako argument "adres"  
PC ← PC + 3, jeśli 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. Następnie 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 następuje 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 podać określone działanie w zależności od rodzaju klawisza, należy przechowywany kod klawisza porównać z konkretną liczbą.

czekaj:

```
MOV A, klawisz ;pobranie kodu wciśniętego klawisza
CJNE A, #65, sprB ;czy wciśnięto klawisz "A" (65 – kod "A")
..... ;tak to wykonuj te instrukcje
```

sprB:

```
CJNE A, #66, sprC ;nie to czy wciśnięto klawisz "B"
..... ;tak to wykonuj te instrukcje
```

sprC:

```
CJNE A, #67, sprD ;nie to czy wciśnięto klawisz "C"
..... ;tak to wykonuj te instrukcje
```

sprD:

```
CJNE A, #68, czekaj ;nie to czekaj na kolejne naciśnięcie
                  klawisza
                  ;tak to wykonuj te instrukcje
                  .....
```

CJNE Rn, #dana, rel

- rejestr Rn (R0...R7) zostaje porównany z argumentem bezpośrednim, jeżeli nie są zgodne zostaje wykonany skok
PC ← PC + 3, jeśli Rn <> dana, to PC ← PC + rel gdzie
n = 0...7

dodatkowo: C ← 0 gdy Rn ≥ dana, lub C ← 1 gdy Rn < dana

- kod: 1 0 1 1 0 n2 n1 n0 gdzie n2 n1 n0 określają jeden z rejestrów R0...R7 stąd kody: B8h...BFh

cykle: 2 bajty: 3 (kod instrukcji + dana + przesunięcie "rel")

przykład:

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

nie_100:

```
CLR P3.0 ;nie równe to zeruj pin 0 portu P3
.....
```

.....

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

CJNE @Ri, #dana, rel

- porównywana jest zawartość komórki w wew. RAM której adres znajduje się w rejestrze Ri (R0 gdy i=0, lub R1 gdy i=1) z argumentem bezpośrednim. Jeżeli się różnią to następuje skok.
PC ← PC + 3, jeśli (Ri) <> dana, to PC ← PC + rel gdzie
i = 0, 1

dodatkowo: C ← 0 gdy (Ri) ≥ dana, lub C ← 1 gdy (Ri) < dana

- kod: 1 0 1 1 0 1 1 i gdzie i=0, 1 stąd kody: B6h, B7h

cykle: 2 bajty: 3 (kod instrukcji + dana + przesunięcie "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 się 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 polecenie 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 zależności od typu argumentu rozróżnia się dwa typy instrukcji, oto one.

DJNZ Rn, rel

- zmniejszona zostaje zawartość podanego rejestru Rn (R0...R7) o jeden, a następnie jeżeli nie jest równa zero, to następuje skok.
PC ← PC + 2, Rn ← Rn - 1, gdzie n = 0...7
jeżeli Rn < 0, to PC ← PC + rel
- kod: 1 1 0 1 1 n2 n1 n0 gdzie n2 n1 n0 określają jeden z rejestrów R0...R7 stąd kody: D8h...DFh
- cykle: 2 bajty: 2 (kod instrukcji + przesunięcie "rel")
- przykład: sekwencja instrukcji
CLR A
MOV R7, CLR A #26
- dodaj:
ADD A, #1
DJNZ R7, dodaj
-

jest równoważna poleceniu

ADD A, #26

co w obu przypadkach powoduje dodanie do akumulatora liczby 26.

DJNZ adres, rel

- zmniejszona zostaje zawartość komórki pamięci wew. RAM o podanym bezpośrednio adresie o jeden, a następnie 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 poprzednim przykładzie.
CLR A
MOV B, #26
- dodaj:
ADD A, #1
DJNZ B, dodaj
-

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

NOP

- ang. "No Operation", nie rób nic
- jest to instrukcja, w wyniku której nie zmienia się stan procesora, z wyjątkiem licznika rozkazów którym po pobraniu tej instrukcji jest zwiększany o jeden.
- kod: 0 0 0 0 0 0 0 0 00h
- cykle: 1 bajty: 1
- przykład: sekwencja instrukcji
MOV A, B
NOP
MUL A, B
- zajmie procesorowi o 1 cykl maszynowy więcej niż sekwencja
MOV A, B
MUL A, B

ale efekt działania będzie taki sam w obu przypadkach.

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

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

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

DL1 EQU 78h

oznacza że w dalszej części programu przez skrót "DL1" należy rozumieć liczbę szesnastkową 78h (120 dziesiętnej). I tak kompilator oraz

"ręczniacy" powinni tłumaczyć (rozumieć) te 3 litery "DL1" jako liczbę 120. I tak jeżeli w dalszej części programu, już w samym kodzie pojawi się np. instrukcja:

MOV DL1, A

oznacza to będzie to samo co instrukcja

MOV 78h, A

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

MOV B, #DL1

co będzie tożsame jako

MOV B, #78h

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

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

SNAKE EQU 0F005h

Wykonanie w dalszej części programu sekwencji:

MOV DPTR, #SNAKE ;załaduj adres do wskaźnika danych

MOV A, #55h ;w liczbie 55h sąsiednie bity są różne: 01010101

MOVX @DPTR, A ;i zapisz do rejestru pod wskazany adres

spowoduje zamierzony efekt. Pierwszą linię można zapisać oczywiście jako:

MOV DPTR, #F005h

.....

.....

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

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

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

INCLUDE ADD32.INC

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

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

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

Też to potrafisz

z ogólnie przyjętą zasadą (jak w całym assemblerze) w czterech różnych postaciach:

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

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

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

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

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

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

zapisanych dziesiętnie.

Czyli można zapisać ten ciąg w inny sposób np.:

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

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

```
bigtab DW 1234h, 65535, 1010101001010101b, 1, 0
```

```
longtab DD 12345678h, 10000234, 0101010b, 1000000000000000000001b
```

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

```
MOV DPTR, #bigtab
```

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

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

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

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

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

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

Na dyskiecie AVT-2250/D znajduje się taki zbiór definicyjny o nazwie 8052.DEF warto go sobie obejrzeć, aby stwierdzić zgodność informacji przedstawionych przeze mnie w niniejszym artykule. Zbiór taki można oczywiście modyfikować, lecz na wstępnym etapie nauki programowania radzę tego nie robić, a przynajmniej zrobić jego wierną kopię przed takimi eksperymentami.

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

i wyszukiwanie błędów, a o modyfikacjach nie wspomnę. Przykład komentarza:

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

Tekstem pochyłym zaznaczono komentarz.

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

```
ORG 8000h
```

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

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

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

```
etyk01:
```

```
etyk02:
```

```
dodaj:
```

```
przykład:
```

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

Program monitora ("BIOS")

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

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

Skończył się więc ten temat, przyszła odpowiednia pora na wyjaśnienie sobie podstawowego pytania: "Co tak naprawdę siedzi w tym "monitorze" prócz funkcji usługowych które już znacie – czyli operacji umożliwiających np. ładowanie (LOAD), podglądanie (EDIT) i uruchamianie programów (JUMP)".

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

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

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

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

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

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

HEXASCII (0235h)

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

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

Procedurę można też wywołać w postaci :

LCALL 0235h
co przyniesie taki sam skutek. "Ręczniacy" mogą sobie od ręki przetłumaczyć na język maszynowy te polecenie jako ciąg bajtów: " 02, 02, 35h " – pierwszy bajt to kod instrukcji LCALL, dwa następne to adres procedury HEXASCII. Tak samo można postępować z każdym innym wywołaniem, jednak w przypadku "komputerowców" ze względu na czytelność programu, należy używać nazw słownych.

A2HEX (024Eh)

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

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

GG_MM_SS ,gdzie GG – pozycja godzin,
;MM – pozycja minut
;SS – pozycja sekund

(podkreślenie oznacza wygaszoną pozycję wyświetlacza)

np. "12 45 00" (godz 12:45 i 00 sekund) należy wykonać sekwencję:

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

DPTR4HEX (025Fh)

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

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

CLS (0274h)

- czyści całe pole wyświetlacza (wstawia spacje na pozycje DL1...8)
- adres wywołania: 0274h
- we: bez parametrów
- wy: czyści wyświetlacz
- traci: R0 i R7 z aktualnie ustawionego zbioru
- przykład:
LCALL CLS ;wyczyszczenie wyświetlacza

Sławomir Surowiński

Lekcja 5

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

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

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

Dla ułatwienia dodatkowo wszystkie linie ponumerowano liczbami w nawiasach.

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

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

- wprowadzenie z klawiatury minut
- wprowadzenie z klawiatury sekund – mamy czas załączenia
- wybór przez użytkownika końcówki procesora (np. z portu P1) sterującej urządzeniem
- oczekiwanie na rozpoczęcie odliczania. Zakładamy że polaryzacja sygnału sterującego będzie ujemna.
- po naciśnięciu odpowiedniego klawisza (umownie nazwanego jako START) przez

operatora timer zaczyna odliczanie w tył do zera z wyświetleniem pozostałego czasu. Założmy że w tym miejscu programu istnieje (po wciśnięciu innego klawisza) dodatkowo możliwość ustawienia innego czasu, np. w przypadku kiedy pomyliliśmy się przy wpisywaniu czasu włączenia.

- po zakończeniu odliczania urządzenie powinno zostać wyłączone, a program powinien powrócić do punktu e) dając dwie możliwości:
 - rozpoczęcia ponownego odliczania po wciśnięciu klawisza START lub
 - powrotu do punktu a) celem poprawienia (zmiany) nastaw minut i sekund.

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

W podanym niżej przykładzie wykorzystane będą pewne, nie omówione wcześniej rejestry procesora, które modyfikowane są automatycznie przez cały czas działania komputerka.

Modyfikacje te odbywają się „w tle”, tzn. że pogram monitora jest skonstruowany tak, że nawet podczas działania naszego programu z przykładu, wspomniane modyfikacje rejestrów odbywają się nieprzerwanie. Dzieje się tak za sprawą układu licznikowego procesora oraz systemu przerwań. Sprawa jest nieco skomplikowana, a ponieważ będzie omówiona w kolejnych lekcjach, nie będziemy jej teraz szczegółowo omawiać.

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

Otóż program monitora do działania potrzebuje kilku rejestrów, które fizycznie znajdują się w obszarze wewnętrznej pamięci RAM

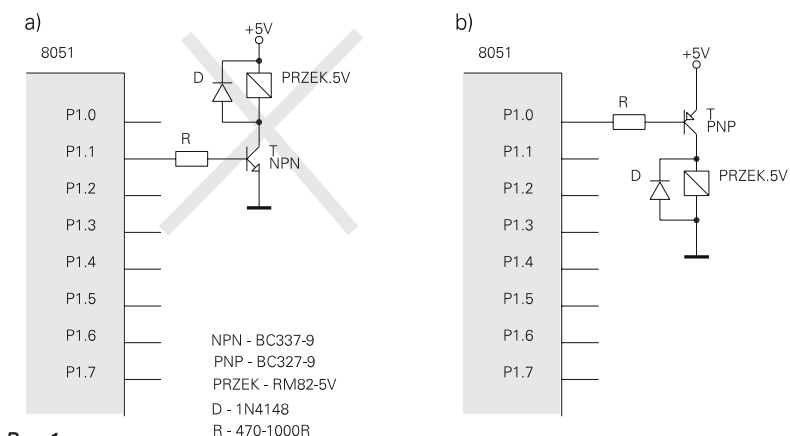
procesora, przeznaczonej do dowolnego wykorzystania przez użytkownika. Oto kilka z nich, wykorzystane w naszym programiku:

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

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

| Bit: | 7-6-5-4-3-2-1-0 |
|-----------|-----------------------------|
| Segmenty: | H G F E D C B A (msb...lsb) |
| Liczba | Binarnie Znak |
| 63 | 00111111 – cyfra '0' |
| 6 | 00000110 – cyfra '1' |
| 91 | 01011011 – cyfra '2' |
| 127 | 01111111 – cyfra '8' |
| 111 | 01101111 – cyfra '9' |
| 119 | 01110111 – litera 'A' |
| 124 | 01111100 – litera 'B' |
| 94 | 01011110 – litera 'D' |
| 64 | 01000000 – znak '-' (minus) |
| 118 | 01110110 – litera 'H' |

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



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

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

BLINKS (adres: 71h) – bajt, którego poszczególne bity odpowiadają za atrybut wyświetlanego na displeju znaku. Ustawienie np. najstarszego bitu w tym słowie powoduje że zapisany do rejestru DL8 znak będzie migotał. Częstotliwość migania określona jest wewnętrznie przez program monitora na 2Hz i oczywiście można ją zmieniać, lecz sposób tej sposob na to nie jest tematem tego artykułu. Przyrządkowanie poszczególnych bitów pozycjom wyświetlacza jest następujące.

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

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

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

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

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

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

Linia (6) powoduje ustawienie wszystkich końcówek portu P1 w stan wysoki, co w naszym załączeniu oznacza wyłączone urządzenie zewnętrzne. Komuś w tym miejscu może wydawać się, że bardziej logiczne byłoby wyzerowanie końcówek i sterowanie np. przekaznikiem za pomocą tranzystora NPN (jak pokazano na rys.1a) lecz takie rozumowanie może w pewnych przypadkach być błędne. Jak wiadomo w prawdziwych zastosowaniach wszelkie stany przejściowe szczególnie układów wykonawczych są niepożądane. Tak też może się zdarzyć w przypadku sterowania przekazywaniem z rys.1a, bowiem procesor po włączeniu zasilania przez krótką chwilę (kilka do kilkuset milisekund) znajduje się w stanie resetu, a końcówki jego portów znajdują się w stanie wysokiej impedancji. W takim przypadku sterowany przełącznik w sposób jak na rys.1a zostałby za-

łączony na krótko po włączeniu zasilania, układu, aż do momentu planowanego wyłączenia na początku programu instrukcją np. MOV P1, #0 – etykieta „pocz:”.

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

“ – – – –F”

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

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

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

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

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

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

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

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

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

Następnie w linii (27) numer wyjścia zapamiętany zostaje w rejestrze B, gdzie poczeka na „swoją chwilę”.

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

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

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

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

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

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

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

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

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

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

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

W tym miejscu zamiast tych dwóch instrukcji można by użyć polecenia CJNE R4, #0, niezer, ale wtedy między etykietę “niezer” a instrukcję kall DECACC trzeba wstawić instrukcję: MOV A, R4” zastanów się jednak dlaczego? Jeżeli warunek jest spełniony, czyli zawartość licznika sekund (rejestru R4) jest różna od zera, następuje skok do miejsca programu oznaczonego przez etykietę “niezer”.

Też to potrafisz

Tutaj w linii (53) wywołany zostaje podprogram, którego zadaniem jest dekrementacja zawartości akumulatora, który przecież zawiera to co przed chwilą było w R4, czyli licznik sekund. Dlaczego użyto podprogramu, a nie np. instrukcji DEC A, która przecież także zmniejsza zawartość rejestru (akumulatora), a no dlatego, że instrukcja ta dekrementuje rejestr w sposób naturalny w kodzie szesnastkowym, czyli np.

59, 58, 57, 56, 55, 54, 53, 52, 51, 50, jak dotąd jest wszystko w porządku, ale następne zmniejszenie rejestru nie spowoduje ustawienia sekund na liczbę 49 ale na 4Fh!, a tego nie chcielibyśmy wyświetlić na displayu jako pozostałych sekund. Gdyby zatem użyć tej pojedynczej instrukcji, to nie tylko odczytu na wyświetlaczu byłyby bezsensowne, ale i 10 sekund w naszym timerku trwałoby aż 16 sekund!

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

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

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

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

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

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

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

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

19:00 (min:sek)

to po zmniejszeniu o 1 sekundę licznik powinien wskazywać

18:59 (min:sek)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

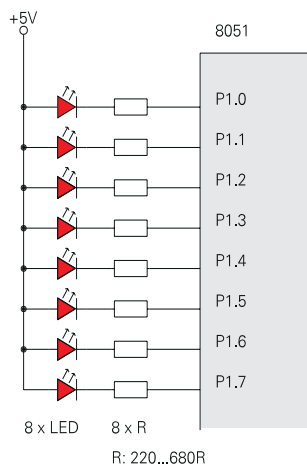
Pierwsza procedura WAITSEK opiera się na zasadzie sprawdzenia dwukrotnej zmiany stanu najmłodszego bitu w słowie specjalnym monitora BSW, opisanym na początku lekcji. Dwukrotna zmiana fazy (stanu) tej flagi wyznacza nam okres 1 sekundy, przy założeniu, że podprogram jest wykonywany cyklicznie. Zastanów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

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

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

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

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



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

Proponuję jako zadania wykonać następujące ćwiczenia.

Zadanie 1

Zaprojektować i użyć zamiast symbolów "F" i "o" inne wskazujące na stan pracy sterowanego urządzenia.

Zadanie 2

Wprowadzić do programu obsługę i rozpoznawanie klawisza "1" jako kończącego pro-

gram napisem "End". Nie zapomnijcie o pętli nieskończonej typu

stop: SJMP stop

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

Zadanie 3

Zmodyfikować tabelę wyjść "tab_wyj", tak aby przy wymienionych poniżej nastawach wyboru wyjścia (0...7) możliwe były odpowiadające im, a wypisane obok kombinacje załączenia wyjść portu P1. Dodatkowo proponuję do portu P1 dołączyć 8 diod LED

z włączonymi w szereg opornikami (180...470 omów) i sprawdzić efekt swojej pracy, obserwując stan wyjść po uruchomieniu tajmera. Przypominam o prawidłowym włączeniu każdej z diod LED w sposób jak na **rysunku 2**.

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

Sławomir Surowiński

Listing programu

| | | | |
|------|-------------|-----------------------|--|
| | CPU | '8052.def' | |
| | | | ;Lekcja nr 5 szkoły mikroprocesorowej w EdW |
| | | | ;program prostego minutnika z ustawianiem czasu |
| | | | ;załączenia zewnętrznego urz'dzenia sterowanego |
| | | | ;poprzez jedna z konców portu P1 |
| | | | ; |
| (1) | 005C | _o equ 01011100b | ;definicja znaku "o" wyświetlacza |
| (2) | 0050 | minuty equ 50h | ;rejestr przechowujący minuty |
| (3) | 0051 | sekundy equ 51h | ;rejestr przechowujący sekundy |
| | | | ; |
| (4) | 8000 | org 8000h | ;początek pamięci zewn.RAM |
| | | | ; |
| | | | ;napisy i znaki informacyjne |
| (5) | 8000 120274 | pocz: lcall CLS | ;wyczyść display |
| (6) | 8003 7590FF | mov P1,#255 | ;wszystkie piny portu OFF |
| (7) | 8006 757840 | mov DL1,#_minus | |
| (8) | 8009 757940 | mov DL2,#_minus | |
| (9) | 800C 757B40 | mov DL4,#_minus | ;napis "—", zachęta do o |
| (10) | 800F 757C40 | mov DL5,#_minus | ;wprowadzenia czasu |
| (11) | 8012 757E40 | mov DL7,#_minus | ;i dodatkowo znacznik "-F" |
| (12) | 8015 757F71 | mov DL8,#_F | ;wylaczenia "F" wyjścia "-" |
| (13) | 8018 1280A4 | lcall WAITSEK | |
| | | | ; |
| | | | ;wprowadzanie czasu - minuty |
| (14) | 801B 757103 | mov blinks,#00000011b | ;miga pozycja minut |
| (15) | 801E 75F001 | mov B,#1 | ;na pozycji nr 1 displaya |
| (16) | 8021 1203A7 | lcall GETACC | ;pobranie liczby minut 0..99 |
| (17) | 8024 F550 | mov minuty,A | ;i zapamiętanie jej w rejestrze |
| (18) | 8026 | | |
| | | | ; |
| | | | ;wprowadzanie czasu - sekundy |
| (19) | 8026 757118 | mov blinks,#00011000b | ;miga pozycja sekund |
| (20) | 8029 75F004 | mov B,#4 | ;na pozycji nr 4 displaya |
| (21) | 802C 1203A7 | lcall GETACC | ;pobranie liczby sekund 0..59 |
| (22) | 802F F551 | mov sekundy,A | ;i zapamiętanie jej w rejestrze |
| | | | ; |
| | | | ;wybor wyjścia - pinu portu P1 |
| (23) | 8031 757140 | mov blinks,#01000000b | ;miga pozycja numeru wyjścia |
| (24) | 8034 75F007 | mov B,#7 | ;na pozycji 7 displaya |
| (25) | 8037 120379 | lcall GETDIGIT | ;pobierz numer pinu P1: 0..7 |
| (26) | 803A 5407 | anl A,#7 | ;dla bezpieczeństwa ważne tylko 3 lsb |
| (27) | 803C F5F0 | mov B,A | ;i przechowanie numeru wyjścia |
| (28) | 803E 757180 | mov blinks,#10000000b | ;miga znacznik załączenia |
| | | | ; |
| | | | ;czekanie na rozpoczęcie odliczania |
| (29) | 8041 1202C5 | wait: lcall CONIN | ;czekanie na klawisz |
| (30) | 8044 B40D57 | cjne A,#klaw_OK,nieok | ;start - klawisz OK |
| | | | ; |
| | | | ;te instrukcje będą wykonane gdy wcisnięto klawisz startu odliczania |
| (31) | 8047 1280A4 | lcall WAITSEK | ;czekanie sekundy |
| (32) | 804A 757A40 | mov DL3,#_minus | |
| (33) | 804D 757104 | mov blinks,#00000100b | ;miga kreska między (mm-ss) |
| | | | ; |
| (34) | 8050 9080CC | mov DPTR,#tab_wyj | ;pobierz adres tabeli wyjść |
| (35) | 8053 E5F0 | mov A,B | ;odtworzenie numeru wyjścia |
| (36) | 8055 93 | movc A,@A+DPTR | ;pobranie maski wyjść z tabeli |

Też to potrafisz

```

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

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

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

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

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

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

;-----

(71) 80A4           WAITSEK:                ;procedura czekania 1 sek.
(72) 80A4 C0E0      push    Acc
(73) 80A6 E570      czek1:  mov     A,bsw          ;pobranie slowa stanu monitora
(74) 80A8 20E0FB    jb      Acc.0,czek1          ;czekanie na wyzerowanie bitu 2Hz
(75) 80AB E570      czek2:  mov     A,bsw
(76) 80AD 30E0FB    jnb     Acc.0,czek2          ;a potem na jego ustawienie
(77) 80B0 D0E0      pop     Acc
(78) 80B2 22        ret

;-----

(79) 80B3           DECACC:                ;dekrementacja Acc z korekcja dziesietna
(80) 80B3 7003      jnz     niero
(81) 80B5 7499      mov     A,#99h
(82) 80B7 22        ret
(83) 80B8 C3        niero:  clr      C
(84) 80B9 9401      subb    A,#1
(85) 80BB C0E0      push    Acc
(86) 80BD 540F      anl     A,#0Fh
(87) 80BF B40F07    cjne    A,#0Fh,nie0F
(88) 80C2 D0E0      pop     Acc
(89) 80C4 54F0      anl     A,#0F0h
(90) 80C6 4409      orl     A,#9
(91) 80C8 22        ret
(92) 80C9 D0E0      nie0F:  pop     Acc
(93) 80CB 22        ret

;-----

(94) 80CC FEFDFBF7  tab_wyj db      11111110b,11111101b,11111011b,11110111b
;wyj: 0,1,2,3
(95) 80D0 EFD0BF7F      db      11101111b,11011111b,10111111b,01111111b
;wyj: 4,5,6,7

;-----

(96) 80D4           END

```




W kolejnej części naszego cyklu zapoznamy się z pozostałymi podprogramami zawartymi w monitorze komputerka edukacyjnego AVT-2250.

W dalszej części artykułu postaram się wyjaśnić znaczenie kilku ważnych dla działania monitora rejestrów. Na końcu jak zwykle proponuję kolejną lekcję szkoły mikroprocesorowej, tematem dzisiejszej będzie konwersacja liczb szesnastkowych na dziesiętne.

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

TEXT (0285h)

- wyprowadza na wyświetlacz znaki począwszy od pozycji DLx podanej w rejestrze B aż do kodu „pustego” – 00h. Kody muszą znajdować się w pamięci programu, a adres ich początku przed wywołaniem procedury powinien znajdować się w rejestrze DPTR. Jeżeli kolejne kody (bajty) opisują znaki np. cyfry, litery: „A”...„F”, „P”, „L”, oraz inne zgodnie z zasadą tworzenia własnego znaku podaną w poprzednim odcinku szkoły mikroprocesorowej, a cały ciąg bajt kończy się liczbą 0, to w efekcie podprogram powoduje wypisanie na wyświetlaczu pseudo-tekstu.

- adres wywołania: 0285h
- we: DPTR wskazuje początek ciągu bajtów w pamięci programu, B – pozycja 1 znaku na wyświetlaczu (1...8)
- wy: wypisuje znaki na wyświetlaczu zgodnie z zasadą opisaną wcześniej
- traci: rejestry DPTR, A i R0
- przykład: powiedzmy że chcemy wypisać pseudo-napis informujący o błędzie, np. „Error”. W pierwszej kolejności dobrze jest zdefiniować trzy znaki (litery), a więc „E”, „r” i „o” opierając się na matrycy wyświetlacza 7-segmentowego. Korzystając z zasady tworzenia znaków opisaną w lekcji 5, definiujemy:

```
_E equ 01111001b ;litera „E” (segmenty B, C i kropka zgazzone) – 79h
_r equ 01010000b ;litera podobna do małego „r” – 50h
_o equ 01011100b ;litera podobna do małego „o” – 5Ch
```

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

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

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

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

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

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

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

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

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

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

DELAY (0295h)

- wykonanie tego podprogramu zajmuje procesorowi określony czas, którego wielkości można określić podając wartość rejestru A (akumulatora przed wywołaniem procedur, zgodnie z zasadą: *czas wykonania = wartość z Acc * 1,95 ms (milisekundy)* – około. Skoro więc do akumulatora można wpisać dowolną liczbę z zakresu 1...255 (nie zalecam w tym przypadku zera) to w efekcie działanie podprogramu DELAY można porównać z wywołaniem celowego opóźnienia o czasie trwania w zakresie około 2...500 ms, czyli od ułamków do prawie do 1/2 sekundy.
 - adres wywołania: 0295h
 - we: Acc – opóźnienie * 1,95 ms
 - wy: j/w
 - traci: Acc
 - przykład:
- ```

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

```

co w programie wywoła opóźnienie około 1 sekundy.

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

## CONIN (02C5h)

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

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

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

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

- adres wywołania: 02C5h
- we: czekanie na klawisz aż do skutku
- wy: w Acc kod naciśniętego klawisza zgodnie z tabelą powyżej

- przykład: powiedzmy że czekamy na klawisz „OK” lub na „4”, poniższy przykład pozwoli nam rozstrzygnąć, który z nich został wciśnięty. Zdefiniujemy na początku kod klawisza OK jako:

```

klaw_OK EQU 13

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

czy4: CJNE A, #4, czekaj ;czy klawisz '4', nie to czekaj dalej

 tu dalsze instrukcje programu właściwe dla klawisza '4'

```

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

Linie tę można oczywiście zapisać jako

```

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

```

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

## GETDIGIT (0379h)

- podprogram służy do wprowadzenia z klawiatury cyfry kodu szesnastkowego (0..9.A..F) z jednoczesnym wypisaniem jej na wyświetlaczu na pozycji podanej w rejestrze B przed wywołaniem procedury. Praktycznym zastosowaniem jest wprowadzanie przez użytkownika programu danych liczbowych w postaci szesnastkowej (lub jak się to okaże za chwilę także dziesiętnej) w sytuacji gdy zachodzi taka potrzeba, np. przy podawaniu adresu obszaru w zewn. pamięci RAM procesora.

- adres wywołania: 0379h
- we: B – numer pozycji na wyświetlaczu (1...8)
- wy: wprowadzona cyfra (0...15) 0...9, A...F
- traci: rejestr R0
- przykład: niech w naszym programie zajdzie potrzeba wprowadzenia liczby graczy, dzięki naszemu podprogramowi można to osiągnąć elegancko za pomocą kilku poniższych linii:

```

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

```

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

```

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

```

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

## GETACC (03A7h)

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



Dodatkowo można określić pozycję na wyświetlaczu od której ma być wypisana wprowadzona liczba.

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

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

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

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

## GETDPTR (03B9h)

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

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

```
wypelniaj:
MOV A, B
MOVX @DPTR, A
INC DPTR
MOV A, DPL
CJNE A, DPL, wypelniaj
MOV A, DPH
CJNE A, DPH, wypelniaj
```

koniec\_wypelniania:

```
.....
.....
```

Przykład byłby krótszy, gdyby istniała instrukcja 8051 typu:

```
CJNE Rn, adres, ofset
```

zastanów się dlaczego i spróbuj uzupełnić program o niezbędne komunikaty.

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

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

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

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

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

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

Dla porządku powiem że przed rozpoczęciem zabawy z portem szeregowym i wywołaniem związanych z nim podprogramów monitora, powiem że będziemy korzystać z typu transmisji który można określić następującymi cechami:

- **asynchroniczna**: oznacza to że urządzenie nadawcze jak i odbiorcze muszą mieć ustaloną tę samą szybkość transmisji, czyli że w takim samym tempie muszą odbierać i nadawać kolejne bity danych. Niestety aby tak było najprościej jest ustawić ręcznie szybkość transmisji w sposób ręczny. W przypadku naszego komputerka najłatwiej jest tego dokonać za pomocą funkcji monitora „BAUD” – klawisz „B”. W „przyrodzie” przyjęto pewne standardowe szybkości transmisji szeregowy asynchronicznej, a mianowicie:

1200, 2400, 4800, 9600, 19200, 38400, 57600 bodów. Liczby te oznaczają ilość przesyłanych bitów w ciągu jednej sekundy, w żargonie komputerowców nazywa się je jednostką „bod” (ang. „baud”). Profesjonaliści na prezentacjach mówią o „bitach na sekundę” (B/sek) lub o kilobitach / sekundę.

Zakres standardowych prędkości jest szerszy, tak w dół jak i górę, lecz my w naszej praktyce będziemy używać podanych wyżej, co w zupełności wystarczy każdemu „zjadaczowi chle...”, chyba „zjadaczowi kompute...”, sami zresztą dokończcie.

- w typowej transmisji zakładamy, że dana jest liczba 8-bitowa (można zatem przysyłać znaki z zakresu 0...255) dodatkowo występuje tzw. bit startu (logiczne „0”) oraz bity stopu, u nas będzie to jeden (logiczne 1). Tak jak wspomniałem wcześniej o szczegółach transmisji dowiemy się później, toteż proszę się nie denerwować chwilowym brakiem wiedzy.

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

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

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

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

| prędkość transmisji | wartość TH1 (8Dh) |
|---------------------|-------------------|
| 1200                | D0h               |

## Też to potrafisz

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

W praktyce modyfikacji rejestru można dokonać za pomocą instrukcji:  
MOV TH1, #baud  
gdzie za wyraz „baud” podstawiamy wybraną z tabeli powyżej wartość, np.  
MOV TH1, #FDh ; ustawienie prędkości transmisji na 19200 bit/sek.

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

Uwaga, rejestr ten nie jest standardowym rejestrem specjalnym SFR procesora 8051, został on zaimplementowany przeze mnie w trakcie tworzenia programu monitora dla komputerka AVT-2250, toteż jeżeli, ktoś np. postanowi sam zaprogramować sterowanie transmisją poprzez port szeregowy, powinien się z tym liczyć. Wnikliwi czytelnicy z pewnością a tym już wiedzą, a no choćby z informacji o adresie rejestru „overtime” równym 74h, a więc znajdującym się poza obszarem rejestrów specjalnych procesora SFR, mniej wtajemniczonym wskazuje o tym przypominam.

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

### INRS (02A5h)

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

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

MOV TH1, #b9600 ;ustaw prędkość na 9600 bodów
MOV R7, #10 ;10 znaków do odebrania
MOV R0, #50h ;bufor na 10 znaków od adresu 50h

następny:
LCALL INRS ;pobierz bajt z portu
JC blad ;jeżeli wystąpił błąd to skocz do etykiety
MOV @R0, A ;zapamiętaj daną w pamięci
INC R0 ;zwiększ adres wskaźnika bufora
DJNZ R7, nastepny ;jeśli nie to następny bajt do odebrania okej:

MOV DPTR, napis_OK ;komunikat o poprawnym odbiorze
pisz: MOV B, #1 ;od pierwszej pozycji
LCALL TEXT ;wypisz na dyspleju

stop: SJMP stop ;stop programu

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

Należy jeszcze zdefiniować komunikaty o przebiegu transmisji, np.:
```

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

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

### OUTRS (02B9h)

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

następny:

```
LCALL INRS ;czekanie na daną z portu
JC blad ;jeżeli przeterminowanie to skocz
LCALL OUTRS ;jeśli nie to odeslij znak
CJNE A, #27, nastepny ;czy bajt = 27, nie to odbierz następny
SJMP stop ;tak to skok na koniec

blad: MOV DPTR, #napis_ERR ;komunikat o błędzie
MOV B, #1 ;na pozycji 1 dyspleja
LCALL TEXT ;wypisz
stop: SJMP stop ;i zakończ program

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

### INACCRS (02F2h)

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

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



```
LCALL OUTRS ;o wysłanie młodszy pól bajtu
..... ;dalsze instrukcje programu lub „RET”
```

```
tabela DB '0123456789ABCDEF
```

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

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

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

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

#### Dodatkowe rejestry

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

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

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

**DL1...DL8** – rejestry bufora wyświetlanych znaków

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

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

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

#### a) instrukcja

```
MOV DL1, #255
spowoduje zapalenie wszystkich segmentów wyświetlacza DL1 (także kropki)
```

#### b) instrukcje

```
MOV DL1, #0
```

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

spowodują to samo co podprogram CLS – wyczyszczenie wyświetlacza.

#### c) wykonanie instrukcji

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

spowoduje pojawienie się znajomego napisu „HELLO” na wyświetlaczu.

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

#### CNT256 – rejestr licznika wyświetlacza (adres: 77h)

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

```
MOV CNT256,
```

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

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

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

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

```
MOV KLAWSZ, #0
```

po odczytaniu klawisza instrukcją: MOV A, KLAWSZ.

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

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

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

#### OVERTIME – rejestr przeterminowania (adres: 74h)

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

czekaj:

```
MOV A, CNT256 ;testowanie rejestru cnt256
JNZ czekaj ;jeżeli <>0 to testuj dalej
```

```
MOV OVERTIME, #liczba_sekund; załadowanie liczby sekund
```

czekaj2:

```
MOV A, OVERTIME ;testowanie rejestru „overtime”
JNZ czekaj2 ;jeżeli nie równy 0 to testuj dalej
```

## Też to potrafisz

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

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

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

Przykład: wykonując instrukcję

MOV OVERCONST, #10

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

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

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

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

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

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

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

DL 1-2-3-4-5-6-7-8

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

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

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

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

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

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

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

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

bit 5 i 4: **nie używane**

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

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

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

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

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

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

**Sławomir Surowiński**

# Lekcja 6

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

### Zadanie 1

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

Np. niech w DPTR będzie liczba 8A12h, w wyniku wywołania tej procedury powinniśmy otrzymać wynik w postaci dziesiętnej, czyli: 35346. Należy przy tym zauważyć, że o ile ilość cyfr liczby szesnastkowej to cztery, o tyle przy zamianie liczb powyżej 270Fh wynik, czyli postać dziesiętna będzie miała aż 5 cyfr. Wynika z tego że wynik operacji zamiany bę-

dzie składał się z trzech bajtów, bo np. zamieniając liczbę 90ach na postać dziesiętną otrzymamy wynik 37036 dziesiętnie, toteż:

| liczba | liczba   |
|--------|----------|
| HEX    | DEC      |
| 90 AC  | 03 70 36 |

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

Zasada konwersji w naszym przykładzie polega na testowaniu każdego z 16-tu bitów liczby, jeżeli jest on ustawiony (=1) to do wyniku dodawana jest liczba będąca potęgą dwójki, której stopień odpowiada numerowi testowanego bitu, tzn. że dla bitu 0 będzie to  $2^0$  czyli 1, dla bitu 1 będzie  $2^1 = 2$ ,..... itd, aż do bi-

tu 15, gdzie  $2^{15} = 32768$ . Jeżeli zaś testowany bit jest równy 0 to wynik pozostaje bez zmian. Dodatkowo podczas dodawania wykorzystano korekcję dziesiętną, dzięki czemu, wynik ma postać dziesiętną, a o to nam przecież chodziło. Jeżeli ktoś chce może na kartce papieru przeprowadzić konwersję tą metodą, jest to dość proste, należy tylko operować na małych wartościach.

A oto kilkanaście linii, dzięki którym przeprowadzana jest konwersja.

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

```

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

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

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

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

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

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

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

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

jz fini

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

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

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

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

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

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

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

```

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

## Zadanie 2

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

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

Za miesiąc rozwiązanie zadania, tymczasem życzę wesołej zabawy.

Sławomir Surowiński





Po dłuższej przerwie w opisie bloków funkcjonalnych mikrokontrolera 8051, wypełnioną opisem niezbędnych do programowania, instrukcji asemblera, kontynuujemy prezentację pozostałych układów wewnętrznych procesora.

Dzięki temu, że już znacie język maszynowy procesora, a przynajmniej orientujecie się w jego składni i sposobach „panowania” na różnego rodzaju rejestrach kostki, będzie mi łatwiej opisywać elementy procesora, bowiem za każdym razem będę ilustrował opisy, przykładami sposobów programowania takich elementów jak port szeregowy, układy licznikowo-czasowe, czy układ przerwań. Cel takiego podejścia do tematu jest oczywisty, to praktyczne nauczanie Was, drodzy Czytelnicy, łatwego i przyjemnego korzystania z wszystkich możliwości mikrokontrolera 8051.



Zanim przejdę do sedna dzisiejszego odcinka, chciałbym zaanonsować wszystkim zainteresowanym Czytelnikom, że od dzisiejszego numeru EdW uruchamiam

„Kącik pocztowy 8051”, w którym będę odpowiadał na wszystkie zawarte w waszych listach problemy, dotyczące programowania naszego mikrokontrolera. Ze względu na to, że część z ogromnej liczby listów które dostaję będzie wiązała się z wybranymi, prezentowanymi w kolejnym odcinku, problemami, odpowiedzi będą wiązały się z tematami, które już omawialiśmy lub tymi, które akurat są tematem kolejnego odcinka klasy mikroprocesorowej.

W dzisiejszej części zapoznamy się z układem transmisji szeregowej oraz praktycznymi sposobami programowania go i wykorzystywania do własnych celów, w tym także do przesyłania danych pomiędzy komputerem PC lub dwoma komputerami edukacyjnymi.

## Port szeregowy

Mikrokontroler 8051 i pochodne posiadają sprzętowy port szeregowy (w skrócie UART), dzięki któremu możliwe jest wysyłanie i odbieranie informacji w postaci szeregowej, czyli „bit po bicie”. Jak już wiecie z opisu wprowadzeń, który przedstawiłem na początku naszego kursu, procesor posiada dwie dedykowane końcówki które wchodziły w skład portu P3 procesora. Są to:

RXD – (P3.0) wejście szeregowo („Receive data”)

TXD – (P3.1) wyjście szeregowo („Transmit data”)

Jak zapewne pamiętacie, końcówki te mogą być wykorzystywane jako uniwersalne wejścia-wyjścia, dzięki instrukcjom zapisu do portu P3, np.

```
MOV P3, #dana
```

lub indywidualnym sterowaniem każdej końcówki portu np.:

```
SETB P3.0 ;ustawienie „1” na końcówce RXD
```

```
CLR P3.1 ;ustawienie „0” na końcówce TXD
```

Jednak przy wykorzystaniu portu szeregowego, sterowanie końcówkami odbywa się automatycznie (za pomocą CPU), według ustawionych wcześniej przez programistę parametrów przesyłowych. Port szeregowy wysyła i odbiera dane w postaci bajtów (8-bitowych słów danych). Konwersja danej wysłanej lub odebranej przez procesor z postaci bajtu do postaci szeregowej lub odwrotnie, odbywa się automatycznie. Dzięki temu wystarczy wskazać tylko dana którą chcemy wysłać lub czekać na odbiór jej z zewnętrznego urządzenia, także wyposażonego w port szeregowy.

Miejsce w którego wysyła się wspomniane dane – bajty, lub do którego one trafiają po transmisji z zewnątrz jest specjalny rejestr, znajdujący się pod adresem 99h w pamięci wewnętrznej danych procesora w

obszarze rejestrów specjalnych SFR. Rejestr ma nazwę SBUF a zapisywać go można tak samo jak każdy inny rejestr, np. instrukcją zapisu poprzez wskaźnik R1:

```
MOV SBUF, @R1
```

W przypadku, kiedy wcześniej ustawiliśmy parametry transmisji i uruchomiliśmy port szeregowy (o tym jak to zrobić – za chwilę), taki zapis spowoduje automatyczne wytransmitowanie bajtu który wcześniej znajdował się pod adresem wskazywanym przez rejestr indeksowy R1.

W przypadku odbioru danej, po zakończeniu transmisji odebrany bajt informacji będzie automatycznie umieszczony w rejestrze SBUF, a fakt zajścia takiego zdarzenia zostanie zasygnalizowany w programie automatycznie. Dzięki temu będziemy wiedzieć, że w rejestrze SBUF czeka na odczytanie gotowa odebrana dana, z którą można zrobić na co się ma ochotę.

Zanim przejdę do omawiania sposobu sterowania transmisją danych poprzez port szeregowy, pragnę wyjaśnić dwa pojęcia związane z portem szeregowym. Być może niektórzy z Was dokładnie wiedzą o co chodzi, lecz pozostałym Czytelnikom należy się wyjaśnienie zasady samej transmisji szeregowej.

Po pierwsze, należy wiedzieć że istnieją praktycznie dwa sposoby na przysyłanie danych metodą szeregową, są to: transmisja **synchroniczna** i transmisja **asynchroniczna**.

Ponieważ port szeregowy procesora 8051 może pracować w obydwu tych trybach, wyjaśnię na początku o co chodzi i jakie są zasadnicze różnice pomiędzy obiema rodzajami transmisji.

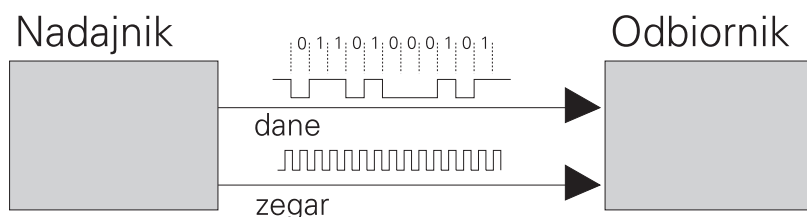
### Transmisja synchroniczna

W tym przypadku dane (informacje) przesyłane są od nadajnika do odbiornika za pomocą dwóch przewodów (nie licząc oczywiście masy). Jednym przesyłane są dane, a drugim generowany jest sygnał zegarowy, w takt którego odbiornik może odebrać informację i stwierdzić, czy nadeszła „1”-ka czy logiczne „0”. Można więc powiedzieć że dane są przesyłane synchronicznie z przebiegiem zegarowym transmitowanym równoległe z danymi, stąd m.in. nazwa rodzaju transmisji.

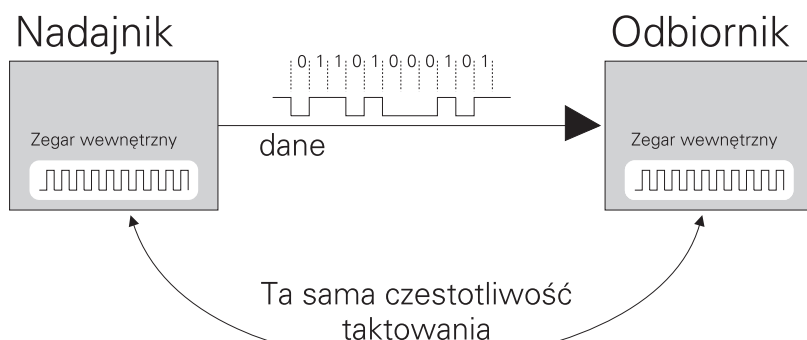
Sytuację tę ilustruje rys.1a. Stan nieaktywny na linii może być umowny, może to być logiczne 0 lub jedynka, umowny jest też sposób generowania sygnału zegarowego, wszystko zależy od przyjętego w układzie rozwiązania. Nie dotyczy to konkretnie mikrokontrolera 8051 i podobnych, lecz tu także ustalono, kiedy transmisja się zaczyna i w jaki sposób generuje się sygnał zegarowy (nazywany czasem „synchronizującym”). O tym jakie zasady obowiązują w przypadku 8051 powiem za chwilę.

## Też to potrafisz

### a) transmisja synchroniczna



### b) transmisja asynchroniczna



Rys. 1. Transmisja synchroniczna i asynchroniczna

#### Transmisja asynchroniczna

W przypadku tego rodzaju transmisji (patrz rys.1b) nie ma oddzielnej linii zegarowej, a dane są przesyłane w takt wewnętrznego sygnału zegarowego, generowanego oddzielnie w nadajniku i odbiorniku. Warunkiem prawidłowego przesłania danych w asynchronicznym sposobie transmisji jest to, aby nadajnik i odbiornik miały ustawioną tę samą częstotliwość wspomnianych sygnałów zegarowych (nazywanych też „taktującymi”). Takie ustalenie prędkości transmisji odbywa się na różne sposoby, z reguły jest to „ręczne” ustalenie przez operatora.

Zauważcie przecież że w naszym komputerku, przed przesłaniem danych z komputera PC można ustawić (za pomocą klawisza „B”-baud) żadaną szybkość transmisji, tak samo można postąpić w portem szeregowym komputera PC, za pomocą komendy DOS'u np.:

```
MODE COM2: 9600, n, 8, 1 {Enter}
```

gdzie szybkość ustalono na 9600 bitów na sekundę (bodów). Pozostałe parametry omówię za chwilę.

Jeżeli zatem urządzenie odbiorcze uczestniczące w przesyłaniu danych wie z jaką szybkością nadawane są dane, to będzie mogło odebrać informację, bez dodatkowej linii zegarowej, jak to miało miejsce w przypadku transmisji synchronicznej. Zaoszczędzimy w ten sposób jeden „przewód”. No tak, ale przecież w mikrokontrolerze 8051 są dwie linie portu szeregowego: nadawania (TXD) i odbioru (RXD). To prawda, nie jest możliwe przesyłanie danych w obie strony po jednej linii w trybie asynchronicznym (oczywiście przy wykorzystaniu sprzętowego portu szeregowego o którym mowa), lecz zauważmy że w przypadku, kiedy np. do naszego komputera dołączymy urządzenie które potrafi tylko odbierać dane, to wystarczy połączyć je tylko 1 przewodem z końcówką TXD procesora (nie zapominając o masie), a linię RXD pozostawić nie wykorzystaną.

Z drugiej strony, wyobraź sobie że jeżeli np. do naszego komputera dołączono oddalony o kilkadziesiąt metrów cyfrowy układ automatycznego termometru, który w określonych odstępach czasu automatycznie przesyła dane dotyczące aktualnej temperatury, to do odbioru tych danych wystarczy dołączyć linię do wejścia RXD procesora.

W obu przypadkach jeżeli chcielibyście wykorzystać transmisję synchroniczną, trzeba by pociągnąć jeszcze jeden przewód między odbiornikiem a nadajnikiem, co zwiększyłoby koszty takiego przedsięwzięcia a jednocześnie skomplikowało instalację urządzenia.

Jakie są zalety a jakie wady obu rodzajów transmisji?...hm, na to pytanie dość trudno jednoznacznie odpowiedzieć, zresztą jedna wyraźna korzyść od razu się rysuje:

zmniejszenie liczby przewodów w przypadku transmisji asynchronicznej.

Istnieją urządzenia które po prostu z definicji potrzebują zewnętrznego sygnału zegarowego, nie tylko do przesyłania danych pomiędzy nim a systemem nadrzędnym, lecz także do poprawnej pracy innych funkcjonalnych bloków systemu. W takich przypadkach transmisja synchroniczna staje się nieodzowna.

Do niedawna panowało mniemanie, że transmisja synchroniczna jest „bezpieczniejsza” od asynchronicznej, a także znacznie szybsza. Tak jednak było dla niedużych połączeń pomiędzy nadajnikiem a odbiornikiem (do kilkudziesięciu centymetrów dla szybkości do kilkunastu MHz). Wynikało to z tego, że jak dotąd urządzenia nadawcze i odbiorcze pracujące w trybie asynchronicznym nie były dość doskonałe, i często przy stosunkowo dużych (choć i tak małych w porównaniu z transmisją synchroniczną) prędkościach transmisji następowały „przekłamanie” i „gubienie” informacji.

W dzisiejszych czasach, kiedy mamy do dyspozycji takie urządzenia jak port szeregowy w kontrolerze 8015 i podobnych, zaawansowane układy UART w komputerach PC, zdolne do transmitowania danych w trybie asynchronicznym z prędkościami nawet do 4Mb/sek. (tak!, czterech megabitów na sekundę), problem ten praktycznie zniknął. Dochodzą do tego jeszcze inne udogodnienia takie jak korekcja błędów oraz kompresja danych, co zwiększa realne szybkości transmisji oraz zwiększa bezpieczeństwo przed utratą często tak cennej informacji.

Jednak w wielu sytuacjach fakt że procesor 8051 potrafi przesyłać danej w sposób synchroniczny, może pomóc w rozwiązywaniu wielu ciekawych zagadnień przy projektowaniu peryferyjnych układów cyfrowych, wykorzystywanych nie tylko w domowym zaciszu. Warto zatem o tym pamiętać.

#### UART w mikrokontrolerze 8051

Teraz, kiedy już wiecie na czym polega różnica pomiędzy transmisją synchroniczną a asynchroniczną możemy przejść do omawiania układu UART w naszym procesorze.

Oprócz rejestru SBUF istnieje dodatkowy rejestr sterujący wszystkimi funkcjami portu, a więc trybem jego pracy, sygnalizowaniem stanu transmisji, czy wreszcie uaktywnianiem odbiornika portu szeregowego. Znajduje się on pod adresem 98h w obszarze wewnętrznej pamięci danych procesora i jest jednym z rejestrów specjalnych SFR. Na sciągawce we wkładce z numeru 11/97 EdW znajduje się opis tego rejestru, jednak przedstawię poniżej nieco dokładniej znaczenie poszczególnych jego bitów.

| adr. bitów | 9Fh | 9Eh | 9Dh | 9Ch | 9Bh | 9Ah | 99h | 98h |      |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|------|
| 98h        | SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI  | RI  | SCON |

**SCON.0 (RI)** – (ang. „Receive Interrupt”) znacznik odebrania przez port szeregowy bajtu, jest jednocześnie znacznikiem zgłoszenia przerwania (przy uaktywnionym systemie przerwań). W przypadku kiedy układ szeregowy mikrokontrolera jest ustawiony na odbiór (odblokowany jest układ odbiornika: bit REN=1), po odebraniu poprawnego znaku z urządzenia zewnętrznego, znacznik ten zostaje automatycznie ustawiony. Zerowanie tego znacznika odbywa się wyłącznie programowo, np. przez instrukcję:

```
CLR RI
```

**SCON.1 (TI)** – (ang. „Transmit Interrupt”) znacznik wysłania przez port szeregowy bajtu, jest jednocześnie znacznikiem zgłoszenia przerwania jeżeli uaktywniono wcześniej układ przerwań. W przypadku kiedy do rejestru SBUF zostanie zapisany znak (bajt) po wytransmitowaniu go przez procesor, bit ten zostaje automatycznie ustawiony (TI=1), co informuje o zakończeniu nadawania znaku przez UART. Podobnie jak w przypadku znacznika RI, znacznik ten jest ustawiany automatycznie a musi być zerowany programowo za pomocą instrukcji np.

```
CLR TI
```

**SCON.2 (RB8)** – (ang. „Receive Bit no. 8”) port szeregowy mikrokontrolera 8051 ma możliwość odbioru i transmisji znaków 9-bitowych – istnieje specjalny tryb pracy UART, o tym za chwilę. W takim trybie w przypadku odbioru znaku z urządzenia zewnętrznego, bit RB8 zawiera właśnie wspomniany 9-ty bit odebranego znaku. Oczywiście 8 pierwszych bitów znaku znajduje się jak poprzednio w rejestrze SBUF.

**SCON.3 (TB8)** – (ang. „Transmit Bit no. 8”) 9-ty bit nadawanego znaku w trybie transmisji z 9 bitami danych. Sytuacja analogiczna do poprzedniej, lecz w tym przypadku aby wysłać 9-bitowy znak poprzez port szeregowy należy najpierw wypisać 9-ty bajt nadawanego znaku do bitu TB8 a potem załadować rejestr SBUF ośmioma młodszymi bitami (bajtem) nadawanego znaku.

**SCON.4 (REN)** – („Receive ENable”) bit uaktywnienia odbiornika transmisji szeregowej. W celu odbioru znaku (oczekiwania na nadejście bajtu z portu szeregowego) należy najpierw wyzerować bit REN, aby odblokować sprzętowy odbiornika znaku zawarty w mikrokontrolerze. W przypadku nadawania znaku bit ten powinien być wyzerowany (REN=0).

**SCON.5 (SM2)** – znacznik maskowania odbioru transmisji. Bit ten może być zmieniany programowo. Ustawienie go (SM2=1) powoduje że odbiornik ignoruje te odbierane znaki, których (w trybie 9-bitowym) 9-ty bit (RB8) jest równy zero (RB=0). W efekcie w takim przypadku nie jest ustawiany znacznik odebrania znaku (RI). Dodatkowo w trybie 8-bitowym (tryb=1) sytuacja jest identyczna kiedy po odebraniu znaku nie został wykryty bit stop'u.

Numeracja trybów ich znaczenie oraz wyjaśnienie bitów „stop'u” już za chwilę.

**SCON.7 (SM0)** oraz **SCON.6 (SM1)** – bity ustalające jeden z czterech trybów pracy portu szeregowego. Oto one:

SM0 SM1 = 00 – tryb 0: Transmisja szeregową synchroniczną, znaki 8-bitowe, taktowane sygnałem zegarowym o częstotliwości Fxtal/12, SM0 SM1 = 01 – tryb 1: Transmisja szeregową asynchroniczną, znaki 8-bitowe, szybkość transmisji może być określana programowo (tryb do pracy np. z PC-tem)

SM0 SM1 = 10 – tryb 2: Transmisja szeregową asynchroniczną, znaki 9-bitowe, szybkość określona jako 1/32 lub 1/64 częstotliwości zegara procesora,

SM0 SM1 = 11 – tryb 3. Transmisja szeregową asynchroniczną, znaki 9-bitowe, szybkość transmisji może być określana programowo (znajduje także zastosowanie przy pracy z PC-tem)

Znaczenie i funkcje poszczególnych trybów są następujące.

## Tryb 0

Jak już powiedziałem w tym synchronicznym trybie przesyłania informacji port szeregowy pracuje nadawając i odbierając znaki 8-bitowe. Zawsze pierwszym nadawanym lub odbieranym bitem jest najmniej znaczący (D0).

Znaki przesyłane są po dwukierunkowej linii P3.0 (RXD). Odbierane są i nadawane za pośrednictwem znanego nam już rejestru SBUF w takt sygnału zegarowego, który generowany jest przez kontroler na linii P3.1 (TXD).

W tym trybie częstotliwość sygnału zegarowego jest stała i jest równa 1/12 (jednej dwunastej) częstotliwości sygnału taktującego procesor. W przypadku użycia obwodu oscylatora procesora z rezonatorem kwarcowym 12 MHz, znaki w tym trybie będą przesyłane z szybkością 1 000 000 bitów / sek. (1 Mb/sek).

Przy nadawaniu znaku obowiązuje zasada, że zapis wysłanego kolejnego bitu znaku w urządzeniu odbiorczym (zewnętrznym np. rejestrze przesuwym) powinien nastąpić przy **narastającym** sygnale zegarowym wytwarzanym na linii TXD.

W przypadku odbioru (REN=1) narastające zbocze sygnału zegarowego powinno powodować przesunięcie zawartości zewnętrznego rejestru przesuwym, z którego odbierane są dane, czyli de facto odczyt odbywa się przy opadającym sygnale przesyłanym linią TXD procesora.

Po odebraniu znaku następuje automatyczne ustawienie znacznika RI, a przy nadawaniu – znacznika TI. Fakt że znaczniki te nie są zerowane automatycznie pozwala programiście na testowanie stanu ich, a co za tym idzie monitorowanie faktu odbioru czy nadania znaku bez potrzeby uruchamiania układu przerwań.

Oto praktyczny przykład ilustrujący tą właściwość.

Procedura nadania znaku z akumulatora bez korzystania z układu przerwań może wyglądać następująco:

OUTRS:

```
mov SBUF, A ;przepisanie zawartości
 ;akumulatora do rejestru UART
```

```
czekaj: jnb TI, czekaj ;testowanie flagi nadania
 ;(czekanie na zakończenie nadawania)
 clr TI ;wyzerowanie flagi nadawania
 ret ;powrót z procedury (podprogramu)
```

Zaś procedur odebrania znaku i umieszczenia go w akumulatorze może wyglądać tak:

INRS:

```
 setb REN ;odblokowanie odbiornika
czekaj: jnb RI, czekaj ;testowanie flagi odbioru
 ;(czekanie na odbiór znaku)
 clr RI ;po odbiorze wyzerowanie flagi
 clr REN ;i zablokowanie odbiornika
 mov A, SBUF ;przepisanie znaku do akumulatora
 ret ;powrót z podprogramu
```

O ile podprogram nadania znaku nie zajmie z reguły więcej niż 10 cykli maszynowych procesora, o tyle sprawa odbioru znaku bez włączonego układu przerwań może być czasochłonna. Popatrzcie przecież, że w przypadku gdy nie będzie nadchodził żaden znak z portu szeregowego, procedura INRS w ogóle się nie zakończy, innymi słowy program może się „zawiesić” w przypadku gdy czekamy na jakiś znak z urządzenia zewnętrznego, a on nie nadchodzi.

Właśnie dlatego użycie układu przerwań przy odbiorze znaków może okazać się bardzo pomocne. Jak tego dokonać omówię w dalszej części cyklu klasy mikroprocesorowej.

Istnieje jednak sposób zabezpieczenia się przed takim „nieskończonym” czekaniem na odbiór znaku bez angażowania często i tak przeciążonego systemu przerwań procesora. Otóż wystarczy oprócz flagi RI sprawdzać stan jakiegoś przyjętego w programie rejestru, który jest automatycznie zmniejszany o jeden np. co 1 sekundę. Założymy że zmniejszanie odbywa się automatycznie w procedurze przerwania generowanej przez jeden z liczników procesora np. T0. W takim przypadku, jeżeli stwierdzimy że rejestr ten (nazywany często rejestrem „przeterninowania”) jest równy zero, kończymy sprawdzanie flagi RI, uznając że nastąpił błąd w odbiorze znaku. Wtedy nasza procedura odbioru znaku będzie wyglądała następująco.

Założmy przy tym że wspomniany rejestr „przeterninowania” nazwalimy np. jako „overtime” i zdefiniowaliśmy jego adres jako np.

overtime EQU 7Fh

czyli 7Fh w pamięci wewnętrznej danych kontrolera. Zakładamy też że w wywoływanej automatycznie co 1 sekundę procedurze przerwania od licznika T0 licznik „overtime” w przypadku stwierdzenia jego wartości jako różnej od zera jest zmniejszany o 1. Wtedy użycie podprogramu:

INRS:

```
 setb REN ;odblokowanie odbiornika
czekaj: jnb RI, jest ;testowanie flagi odbioru
 ;(czekanie na odbiór znaku)
 mov A, overtime ;sprawdzenie rejestru
 ;„przeterninowania”
 jnz czekaj ;jeżeli jeszcze <=0 to czekaj
 ;na znak
 sjmp koniec ;jeżeli =0 to zaniechaj czekania
 ;po odbiorze wyzerowanie flagi
 mov A, SBUF ;przepisanie znaku do akumulatora
 clr REN ;i zablokowanie odbiornika
 ret ;powrót z podprogramu
```

z uprzednim załadowaniem rejestru „overtime” liczbą sekund przeznaczonych na maksymalne oczekiwanie na odbiór znaku poprzez instrukcję:

```
mov overtime, #10 ;10 sekund na odbiór znaku
lcall INRS ;i wywołanie podprogramu odbioru
```

spowoduje oczekiwanie maksymalnie 10 sekund na dobiór znaku z portu szeregowego.

Opisana zasada znajduje zastosowanie w trzech pozostałych trybach pracy portu szeregowego, a więc w trybach 1, 2 i 3.

## Tryby 1, 2 i 3

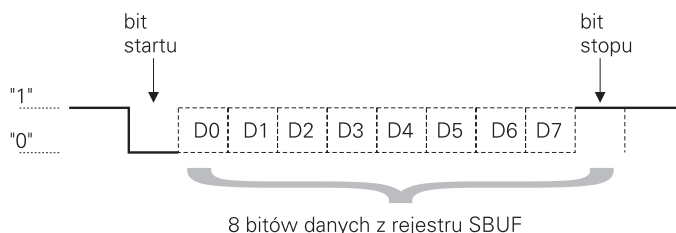
Zanim opiszę szczegółowo wspomniane trzy pozostałe tryby pracy portu szeregowego kontrolera powinienem przedstawić wspólną cechę charakteryzującą te tryby a mianowicie postać przesyłanej asynchronicznej informacji, czyli format przesyłania znaku (8-bitowy lub 9-bitowy).

Ponieważ w trybie asynchronicznym nie istnieje linia przesyłająca sygnał taktujący poszczególne nadawane i odbierane bity, obie strony nadawcza i odbiorcza muszą w jakiś sposób „wiedzieć” o tym że np. w danej chwili nadajnik rozpoczął nadawanie znaku. Wtedy odbiornik

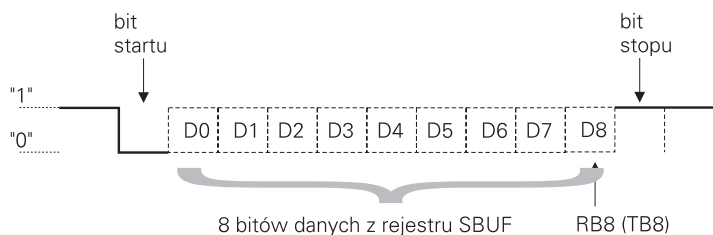


## Też to potrafisz

### a) format danych w trybie 8-bitowym (tryb 1)



### b) format danych w trybie 9-bitowym (tryby 2, 3)



Rys.2. Format znaku w trybie 8-bitowym (a) oraz 9-bitowym (b).

detektując takie zajście będzie, znając oczywiście częstotliwość nadawania znaku przez nadajnik (znając prędkość transmisji), wiedział w jaki sposób odbierać nadawany z zewnątrz znak.

Ustalono, że podczas „ciszy na łączach”, linie portów (RXD – odbioru i TXD – nadawania) są w stanie wysokim. Sygnałem rozpoczęcia nadawania znaku, a z drugiej strony sygnałem konieczności jego odbioru jest pojawienie się tzw. „bitu startu”, czyli niskiego poziomu logicznego na linii (TXD w przypadku nadawania lub RXD – w przypadku odbioru).

Bit startu trwa dokładnie tyle ile powinny trwać (w zależności od szybkości transmisji) pozostałe bity informacji. Na rysunku 2 oznaczono ten bit dokładnie.

Po bicie startu (pamiętajmy, zawsze równy zero!), następują kolejne bity danych. I tak pierwszy transmitowany jest najmłodszy bit (D0) bajtu wpisanego do rejestru SBUF, potem starszy (D1) i tak dalej aż do bitu D7, a w przypadku transmisji 9-bitowej dodatkowo transmitowany jest bit SCON.3 (TB8), po czym następuje bit stopu, który zawsze jest równy „1”.

Pojawienie się bitu stopu kończy nadanie znaku, a po drugiej stronie jego odbiór.

Mechanizm transmisji znaku w trybach 1, 2 i 3 jest taki sam, różna jest tylko liczba bitów danych oraz szybkość transmisji, zgodnie z opisem wcześniej.

Dla przykładu powiem, że choć transmisja 9-bitowa może wydawać się mniej naturalna i niepotrzebna, to jednak jest ona często wykorzystywana do przesyłania danych z tzw. bitem parzystości. Jest to prosty sposób na wyeliminowanie odbioru zafałszowanych danych, kiedy to pomimo, że w określonej, koniecznej chwili nastąpiło wygenerowanie poprawnego bitu startu o raz bitu stopu, to jednak bity danych uległy zniekształceniu, co spowodowało zafałszowanie danych i odbiór niewłaściwego znaku.

Przy korzystaniu z transmisji z komputerem PC można poleceniem MODE ustawić tryb transmisji portu szeregowego komputera na kilka sposobów, a mianowicie:

*Jak się za chwilę przekonasz nasz UART w mikrokontrolerze 8051 potrafi pracować z większymi szybkościami, lecz ustawienie tych szybkości transmisji w PC-cie jest możliwe tylko za pomocą specjalnie utworzonych programów, które są dostępne jako programy terminalowe „shareware”, lub mogą być napisane przez Czytelników, tych którzy potrafią to zrobić. Najprostszym a jednocześnie ogólnie dostępnym programem tego typu jest program pochodzący z pakietu popularnego DOS-owego „Norton Commander’a” pod nazwą: „TERM90.EXE”. O sposobie używania tego programu nie będę się tu rozwodził, zasady powinien znać każdy komputerowiec, a jeżeli nie to radzę się tym programem nieco pobawić.*

MODE COMx: baud, parity, data, stop, retry  
gdzie, poszczególne nazwy oznaczają:

x – określa numer portu szeregowego, do którego dołączony jest UART mikrokontrolera (1 dla COM1, 2 dla COM2, 3 dla COM3 itd.)

baud – określa szybkość transmisji w bitach na sekundę (standardowe najczęściej używane szybkości to: 600, 1200, 2400, 4800, 9600, 19200 bodów czyli bitów/sek)

parity – parametr określający sposób kontroli parzystości. Możliwe wartości to: „o”-odd (nieparzysty) lub „e”-even (parzysty)

data – parametr określający liczbę bitów danych. Wartości typowe akceptowane przez instrukcję MODE to 5...8

stop – liczba bitów stopu. Wartości akceptowane to 1 lub 2 (bity stopu).  
Parametru „retry” nie należy w ogóle podawać.  
W ten sposób można zdefiniować parametry transmisji w komputerze PC dla następujących trybów pracy portu szeregowego mikrokontrolera 8051, np.:

MODE COM2: 9600, n, 8, 1 (1)

dla trybu 1 portu UART w 8051, prędkości 9600 bodów, dołączonego do portu COM2 komputera PC, a w przypadku trybu 9-bitowego (3) :

MODE COM2: 9600, e, 8, 1 (2)

w przypadku badania parzystości bitów danych lub

MODE COM2: 9600, o, 8, 1 (3)

dla nieparzystości.

Najczęściej jednak przydaje się ustawienie portu komputera PC jak w linii (1) oraz korzystanie z trybu 1 UART kontrolera 8051.

W trybie 2 pracy, port szeregowy taktowany jest sygnałem zegarowym o częstotliwości Fxtal/32 lub Fxtal/64 (gdzie Fxtal to częstotliwość rezonatora oscylatora). O tym, która z częstotliwości będzie taktowała port, decyduje stan bitu 7 (SMOD) w rejestrze SFR o nazwie PCON (adres: 87h). Ustawienie tego bitu powoduje podwojenie szybkości transmisji (Fxtal / 32) wyzerowanie – ustawienie taktowania na Fxtal/64.

Jak wspominałem wcześniej w trybach 1 i 3 szybkość transmisji może być określana programowo. W tym przypadku układ transmisyjny taktowany jest za pomocą sygnału przepełnienia licznika T1 układu czasowo-licznikowego.

W kontrolerach 8032/52 do taktowania portu może także posłużyć licznik dodatkowy T2. Jak tego dokonać pokażę na przykładach po opisie sposobów modyfikacji i wartości szybkości transmisji.

Ponieważ wgłębianie się w przebiegi i szczegóły nadawania i odbioru znaków w trybach asynchronicznych jest mało praktyczna, zainteresowanych odsyłam do lektury [1]. Poniżej skupię się jedynie na praktycznych aspektach korzystania z dobrodziejstw portu szeregowego oraz programowania jego rejestrów.

Zanim do tego przejdę zapoznamy się z możliwościami modyfikowania

#### Szybkości transmisji

W trybie 0 szybkość przesyłania danych, jak powiedziałem wcześniej jest niezmienna i wynosi Fxtal/12.

W trybie 2 prędkość transmisji wynosi: Fxtal/32 przy SMOD=1, lub Fxtal/64 przy SMOD=0.

W trybach 1 i 3 prędkość ma się nieco inaczej. W tym przypadku prędkość transmisji określa wzór:

$$n = \frac{\text{Fxtal}}{(256 - \text{TH1}) \times 12 \times \text{dz}}$$

gdzie: dz = 32 w przypadku gdy SMOD = 0 zaś dz = 16 gdy SMOD = 1, zaś TH1 to wartość początkowa 8-bitowego licznika TH1 (starszy bajt T1) pracującego w trybie taktowania portu szeregowego.

Aby wykorzystać standardowe prędkości transmisji znane z komputerów PC oraz innych wyposażonych w asynchroniczny port szeregowy (Commodore, Amiga), czyli: 600, 1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 76800 bit/sek należy zastosować taki kwarc (oscylator) aby po lewej stronie równania otrzymać te wartości przy pamiętaniu że wartość TH1 jest liczbą całkowitą z zakresu 0...255.

I tu wyjaśni się sprawa stosowania trochę dziwnych wartości rezonatorów kwarcowych w niektórych układach wykorzystujących procesory rodziny 8051 i ich port szeregowy.



## Też to potrafisz

Ot chociażby w naszym komputerku edukacyjnym zastosowany rezonator kwarcowy ma wartość 11,0592 MHz, zobaczmy dlaczego i co w efekcie uzyskamy.

Podstawiając do powyższego wzoru, określającego prędkość transmisji danych, wartość oscylatora równą 11059200 Hz oraz zakładając że SMOD = 1 (czyli dz = 16) otrzymamy np. dla TH1 = F4h (244 dziesiątne):  $n = 11059200 / ((256 - 244) \times 12 \times 16) = 4800$  bodów.

Postępując podobnie z innymi wartościami początkowymi licznika TH1 otrzymamy dla podanego rezonatora oraz innych (których częstotliwość rezonansowa dzieli się przez liczbę 12 i 32 (lub 16)) inne standardowe wartości, oto one:

| Fxtal (MHz) | SMOD | TH1 | n (bodów) |
|-------------|------|-----|-----------|
| 11,0592     | 1    | FFh | 57600     |
| 11,0592     | 1    | FEh | 28800     |
| 11,0592     | 1    | FDh | 19200     |
| 11,0592     | 1    | FCh | 14400     |
| 11,0592     | 1    | FAh | 9600      |
| 11,0592     | 1    | F4h | 4800      |
| 11,0592     | 1    | E8h | 2400      |
| 11,0592     | 1    | D0h | 1200      |
| 11,0592     | 1    | A0h | 600       |
| <hr/>       |      |     |           |
| 14,7456     | 1    | FFh | 76800     |
| 14,7456     | 1    | FEh | 38400     |
| 14,7456     | 1    | FCh | 19200     |
| 14,7456     | 1    | F8h | 9600      |
| 14,7456     | 1    | F0h | 4800      |
| 14,7456     | 1    | E0h | 2400      |
| 14,7456     | 1    | C0h | 1200      |
| <hr/>       |      |     |           |
| 1,8432      | 1    | FFh | 9600      |
| 1,8432      | 1    | FEh | 4800      |
| 1,8432      | 1    | FDh | 2400      |
| 1,8432      | 1    | FAh | 1200      |
| 1,8432      | 1    | F4h | 600       |

Jak zapewne się domyślicie wyzerowanie znacznika bitu SMOD w rejestrze PCON (87h) spowoduje zmniejszenie o połowę wszystkich prędkości transmisji podanych w tabeli. I tu uwaga, ponieważ rejestr PCON nie może być adresowany bitowo, czyli nie da się zmodyfikować wspomnianego bitu w tym rejestrze instrukcja np.

```
setb SMOD
```

stąd trzeba użyć innego polecenia, a mianowicie:

```
anl PCON, #7Fh ;aby SMOD = 0
```

lub

```
orl PCON, #80h ;aby SMOD = 1
```

sprawdźcie na kartce papieru lub kalkulatorze.

W tabeli podano najczęściej używane wartości rezonatorów kwarcowych dla zastosowań portu szeregowego 8051. Najbardziej popularną jest wartość 11,0592 MHz, dość rozpowszechniona w handlu, w szczególnych zastosowaniach przydaje się rezonator o Fxtal = 14,7456 MHz, kiedy chcemy aby nasz mikroprocesor konsumował jak najmniej prądu a jednocześnie pracował przy kilku standardowych prędkościach transmisji, powinniśmy użyć kostki w wersji CMOS (80C51/80C31) oraz zastosować rezonator o Fxtal = 1,843200 MHz.

Możliwe jest także w praktyce stosowanie „okrągłych” wartości rezonatorów kwarcowych, lecz w tym przypadku należy liczyć się z błędami szybkości, które jednak przy małych prędkościach transmisji nie mają wielkiego znaczenia dla poprawnej pracy układu UART mikrokontrolera 8051. Wynika to z faktu że błędnie odebrany bit zawsze „leży” poza transmitowanymi 9-cioma czy 10-cioma bitami (nie występuje), które wchodzi w skład transmitowanego w trybie asynchronicznym słowa. I tak np. z zastosowaniem rezonatora kwarcowego Fxtal = 12 MHz możemy śmiało wymieniać dane z komputerem PC lub innym urządzeniem wykorzystującym łącze szeregowe i standardowe prędkości transmisji prędkościami:

## Też to potrafisz

2400 bodów (TH1 = F3h, SMOD = 0), wtedy błąd wyniesie tylko: 0,16%  
4800 bodów (TH1 = E6h, SMOD = 0), wtedy błąd wyniesie także 0,16%

ale już przy prędkości transmisji równej

9600 bodów (TH1 = F9h, SMOD = 1), błąd wyniesie aż 7%, bo aktualna szybkość transmisji będzie mniejsza od założonej i wyniesie w tym przypadku: 8923 body.

W przypadku użycia do taktowania portu szeregowego w kontrolerach 8032/52 licznika T2, prędkość transmisji określona będzie wzorem:

$$n = \frac{F_{xtal}}{(65536 - RLD) \times 16 \times 2}$$

gdzie RLD to wartość początkowa, zapisana przez program użytkownika do pary rejestrów RLDH i RLDL. Jeżeli zastosujemy rezonator kwarcowy o częstotliwości np. 16 MHz to możliwe do uzyskania częstotliwości transmisji będą z przedziału: 7...50000 bitów / sek.

### Praktyczne wskazówki

Jak zatem zaprogramować rejestr sterujący portu szeregowego oraz jak zmusić do pracy licznik którego zadaniem będzie taktowanie transmisji. Oto przykład który pomoże wam zrozumieć ten problem.

### Przykład

Zaprogramujemy UART tak, że będzie pracował w trybie asynchronicznym 8-bitowymi znakami, a taktowany będzie licznikiem T1. Prędkość transmisji określimy np. na 19200 bitów / sek (bodów). Inne wartości transmisji – patrz tabela.

Aha, byłby zapomniany, zakładamy że w systemie zastosowano rezonator kwarcowy o Fxtal = 11,0592 MHz.

Oto sekwencja początkowa uruchamiająca pracę portu:

```
INIT_RS:
 mov SCON, #01000000b ;tryb 1 portu
 orl TMOD, #00100000b ;licznik T1 tryb pracy 8-bitowej
 ;liczy TL1 z automatycznym
 ;wpisywaniem wartości
 ;początkowej
 ;z TH1
 ;ustawienie SMOD = 1
 ;prędkość = 19200 bodów
 ;(tabela)
 setb TR1 ;start licznika T1 (TL1)
 ;i gotowe...
```

Teraz wystarczy użyć procedur które opisałem wcześniej (INRS i OUTRS) aby wysłać dowolny znak poprzez łącze szeregowe. Zmieniając wartość ładowaną do rejestru TH1 (linia: mov TH1, #xx) można w dowolnej chwili zmieniać prędkość transmisji w zakresie zależnym oczywiście od zastosowanego rezonatora kwarcowego.

Analogicznie, opierając się o opis rejestru licznika T2CON (w kostkach 8032/52) można zaprogramować i zmusić do pracy licznik T2, radzę poeksperymentować w domu, oczywiście zaopatrując się wcześniej w odpowiedni mikrokontroler.

No dobrze ale czy można bezpośrednio dołączyć końcówki portu procesora 8051 (TXD i RXD) do np. komputera PC?... absolutnie nie. Potrzebny jest do tego odpowiedni konwerter poziomów napięć. Przykład takiego rozwiązania powinien nasunąć się Wam sam, no tak przecież w naszym komputerku edukacyjnym zastosowaliśmy taki właśnie układ, wykorzystujący popularną przetwornicę w postaci układu scalonego MAX232 lub ICL232.

Więcej na temat portu szeregowego możecie przeczytać w kilku ostatecznych numerach EdW (6, 7, 9 z roku 1997).

Dodatkowe szczegółowe informacje oraz specyficzne cechy układu UART mikrokontrolera 8051 o pochodnych znajdziecie w dodatkowej literaturze [1].

**Sławomir Surowiński**

### Literatura

[1] A. Rydzewski – Mikrokomputery jednoukładowe rodziny MCS-51, WNT1992, 1995

# Lekcja 7

W dzisiejszej lekcji rozwiążę zadania z poprzedniego miesiąca i wyjaśnię sposób wykonywania przytoczonych procedur arytmetycznych przez mikrokontroler 8051. dla ułatwienia będę numerował wszystkie linie w nawiasach tuż obok komentarza.

## 1. Procedura dodawania dwóch liczb 16-bitowych

Pierwszy składnik będzie np. umieszczony w rejestrach DPH i DPL (DPTR)

Drugi składnik dodawania będzie umieszczony w rejestrach B (starszy bajt) i A (młodszy bajt). W wyniku dodawania powstanie liczba w rejestrach B.A oraz dodatkowo przy przekroczeniu zakresu liczb zostanie ustawiony znacznik przeniesienia C.

; C.B.A = DPTR + B.A

ADD16:

```
add A, DPL ;(1) dodanie LSB
push Acc ;(2) przechowanie LSB wyniku na stosie
mov A, B ;(3) przepisanie składnika MSB do Acc
addc A, DPH ;(4) dodanie składnika 2
mov B, A ;(5) przepisanie wyniku do rejestru B
pop Acc ;(6) odtworzenie LSB wyniku
ret ;(7) i koniec (powrót) podprogramu
```

W podprogramie wykonano operację:

|       |     |     |
|-------|-----|-----|
|       | B   | A   |
| +     | DPH | DPL |
| <hr/> |     |     |
| C     | B   | A   |

Popatrzmy na zapis matematyczny. Najpierw dodajemy składniki a prawej strony, czyli A i DPL (linia 1). Potem aby dodać starsze bajty (B i DPH) trzeba było użyć znowu akumulatora, lecz aby to zrobić należało najpierw zachować wynik z Acc, stąd linia (2) – posłużono się stosiem. Następnie przepisano zawartość B do A (linia 3), aby ją dodać za chwilę (linia 4) do MSB drugiego składnika, który przecież znajduje się w DPH. Na końcu przepisano tak powstałą sumę MSB do rejestru B (linia 5) i odtworzono akumulator (linia 6).

A teraz dodatkowe zadanie, poniżej przedstawię alternatywny zapis powyższej procedury, spróbuj się zastanowić, dlaczego oba przykłady w efekcie wykonują to samo?

; C.B.A = DPTR + B.A

ADD16\_2:

```
add A, DPL
xch A, B
addc A, DPH
xch A, B
ret
```

## 2. Procedura odejmowania dwóch liczb 16-bitowych

Odejmowanie dwóch liczb 16-bitowych przeprowadzimy w podobny sposób jak to miało miejsce z dodawaniem, a więc od pary rejestrów B.A odejmiemy zawartość DPTR, wynik pozostanie w B.A

; B.A = B.A – DPTR

SUBB16:

```
clr C ;wyzerowanie znacznika C
subb A, DPL
push Acc
mov A, B
subb A, DPH
mov B, A
```

```
pop Acc
ret
```

prawda że proste! komentarz jest chyba zbędny, poza wyjaśnieniem, dlaczego na początku procedury wyzerowano znacznik przeniesienia C. Otóż instrukcja SUBB procesora 8051 powoduje z definicji odjęcie dwóch składników działania z uwzględnieniem znacznika C. Toteż profilaktycznie należy go wyzerować przez wykonaniem procedury, inaczej w przypadku gdy będzie C=1, wynik będzie obciążony błędem (będzie za mały o 1). Podobnie jak poprzednio procedurę można zapisać jako:

; B.A = B.A – DPTR

SUBB16\_2:

```
clr C
subb A, DPL
xch A, B
subb A, DPH
xch A, B
ret
```

**Uwaga!** W przypadku kiedy B.A będzie mniejsze od DPTR, obliczony zostanie moduł różnicy, a ujemny znak działania będzie sygnalizowany ustawieniem znacznika C!

## 3. Procedura mnożenia liczby 16-bitowej przez 8-bitową

W tym przypadku pomnożymy zawartość rejestru DPTR przez Acc (akumulator). Wynik będzie zatem 24-bitowy, a umieszczony w DPTR. A (DPH. DPL. A)

Zauważmy że mnożenie możemy zapisać jako:

DPH. DPL x Acc = DPL x Acc + DPH x Acc x 100h

Jak widać najpierw trzeba wymnożyć starszy (MSB) oraz młodszy bajt (LSB) rejestru DPTR przez akumulator, a potem odpowiednio dodać składniki. Wykonam zatem działania:

wymnożę DPL x Acc  
mnożę DPH x Acc  
a następnie dodam w kolejności

|       |     |     |     |
|-------|-----|-----|-----|
|       | DPH | DPL | 00h |
| +     |     | B   | A   |
| <hr/> |     |     |     |
|       | DPH | DPL | A   |

i w ten sposób otrzymam wynik. Skorzystam przy tym z instrukcji mnożenia dwóch liczb 8-bitowych. A oto procedura:

MUL24:

```
push Acc ;przechowanie mnożnika na stosie
mov B, DPH
mul AB ;mnożenie DPH x Acc
mov DPH, B ;i zapisanie wyniku – MSB
push DPL ;przechowanie wyniku pośredniego
mov DPL, A ;przepisanie LSB wyniku mnożenia DPH x Acc
pop B ;odtworzenie wyniku pośredniego
pop Acc ;odtworzenie mnożnika
mul AB ;mnożenie DPL x Acc
xch A, B
add A, DPL ;dodanie Acc + DPL
mov DPL, A
jnc less ;czy > FFh
inc DPH ;tak to korekcja na DPH wyniku
less: xch A, B ;odtworzenie Acc
ret ;...i koniec mnożenia, wynik w DPTR. A
```

#### 4. Procedura dekrementacji z korekcją akumulatora

Jak wiemy zastosowanie instrukcji

dec     A

w przypadku kiedy w akumulatorze znajduje się liczba np. 50h da w rezultacie wynik w postaci liczby 4Fh. A co zrobić aby zamiast takiego wyniku otrzymać po „podobną do ludzi” liczbę 49h. Do takiej dekrementacji czyli zmniejszania o 1 liczb zapisanych w kodzie BCD, posłużyć może poniższa procedura, oto ona:

DECACC:

```

jnz nie0 ;(1) sprawdzenie czy Acc=0
mov A,#99h ;(2) tak to załadowanie liczby 99h
ret ;(3) i powrót z podprogramu
nie0: clr C ;(4) nie to zeruj wskaźnik C
 subb A,#1 ;(5) odejmij od akumulatora 1
 push Acc ;(6) i przechowaj wynik na stosie
 anl A,#0Fh ;(7) zamaskuj 4 msb wyniku
 cjne A,#0Fh,nieF ;(8) czy druga cyfra to „F”?

```

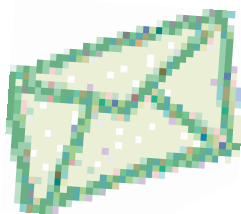
```

pop Acc ;(9) tak to odtwórz wynik
anl A,#0F0h ;(10) i zamaskuj 4 lsb
orl A,#9 ;(11) a na miejsce 4 lsb wpisz cyfrę „9”
ret ;(12) i zakończ podprogram
nieF: pop Acc ;(13) jeżeli druga cyfra to nie „F” to
 ;odtwórz wynik
 ret ;(14) i zakończ podprogram

```

Procedura sprawdza trzy warunki:

- jeżeli w akumulatorze jest liczba 0, to wpisywana jest liczba 99h i procedura zostaje zakończona (linia 1...3)
- jeżeli po zwykłym odjęciu jedynki (linia 5) druga cyfra wyniku jest równa „F” to przeprowadzana jest konwersja wyniku do postaci BCD – linie 9...11. Taka sytuacja ma miejsce gdy liczba wejściowa jest „okrągła” czyli np. 30h, 40h, 50h, 60h, itp.
- jeżeli po odjęciu jedynki druga cyfra wyniku jest z przedziału 0...9 to nie rób nic tylko opuść podprogram – wynik jest w porządku. Sytuacja taka mam miejsce w przypadku gdy liczba wejściowa ma na drugiej pozycji cyfrę z przedziału: 1...9.



## Kącik pocztowy 8051

*Zasypany wieloma listami, dotyczącymi cyklu artykułów poświęconych programowaniu mikrokontrolerów 8051, postanowiłem uruchomić kącik pytań i odpowiedzi, które kierujecie do mnie w swoich listach.*

*Na wstępie chciałem bardzo podziękować za każdy list, zarówno te pochwalne jak i krytyczne. Przyznam że, cieszy mnie bardzo fakt, iż tak wielu z Was zdecydowało się sięgnąć po „mikroprocesory”, a co najważniejsze odnosi już małe ale jak ważne w nauce sukcesy.*

Na początek chciałem szczególnie podziękować: **Rafałowi Majdzie** z Krakowa, **Hubertowi Hawłasowi** z Warszawy, **Zdzisławowi Chmielewskiemu** z Tomaszowa Mazowieckiego, **Robertowi Szymaszkowi** z Bielsko-Białej, **Tomaszowi Jabłonowskiemu** z Moniek, **Jarosławowi Chudobie** z Gorzowa Wlkp.

Specjalnie podziękowania także dla pana **Wiesława Zacharskiego** z Karoliną i Piotrkim.

Chciałbym jednocześnie uspokoić **Konrada Iwaniuka** z Mysłowic, drogi Konradzie prześlij swój komputer pod adresem redakcji AVT z dopiskiem na kopercie „Serwis”, problem zostanie rozwiązany.

Wyrazy uznania dla **Przemka Pietrzyka** z Lublina za jego wyczerpujący list z zamieszczonymi, własnoręcznie napisanymi programami – brawa „ręczniakom”.

Gorąca prośba ode mnie: piszcie proszę swoje adresy nie tylko na kopertach ale także w treści korespondencji. Przyznam że często koperty z natury swojej zostają oddzielone o listu i późniejsze dopasowanie obu części jest praktycznie niemożliwe.

No dobrze, przejdźmy zatem do kilku listów.

\*

*...Jestem początkującym studentem telekomunikacji jak również czytelnikiem gazety EdW. ... Nawiązując do sprawy z zaciekawieniem śledzę artykuły dotyczące 8051, lecz zakup kitu jest dość kosztowny i moje pytanie i prośba do Was – czy jest możliwość zakupu samej kostki EPROM wraz z „monitorem”?*

**Rafał Majda z Krakowa**

**Odpowiedź:** No cóż, nie wszystkim dobrze się powodzi, aczkolwiek przyznam że trochę jestem zaskoczony twierdzeniem że kit komputerka jest kosztowny. Muszą przyznać że AVT poczyniło wszelkie kroki, aby ów zestaw był jak najtańszy. Porównując obecne ceny układów scalonych oraz płytek drukowanych doszedłem wielokrotnie do wniosku, że samodzielny zakup elementów oraz wykonanie (ręczne !, tak ręczne) płytek w sumie daje koszt zbliżony do ceny zestawu. Dochodzi do tego sprawa pozornej prostoty układu, i jak

wynika z waszych listów, częste próby wykonywania układu na własnych PCB kończą się problemami. Podziwiam jednak determinację Czytelników i chęć wykonania wszystkiego samemu (sam taki byłem), dlatego informuję że zaprogramowany EPROM można nabyć za pośrednictwem Działu Handlowego AVT.

\*

*...Jeżeli chodzi o sam układ (elektroniczny) to zastrzeżeń nie mam, ale jeżeli chodzi o zaprojektowaną płytkę, to zastanawia mnie kilka rzeczy: czy umieszczone złącze ARK do zasilania jest tu najlepszym rozwiązaniem... lepiej by było zastosować standardowy wtyk zasilania. nie wiem czym kierował się projektant płytki podłączając w dość dziwnej kolejności linie do złącza PORT, nie wierzę w to że na dwustronnej płytce nie dało się połączyć tego normalnie. Jak już wspominałem mam płytki przykręcone jedna nad drugą i prawdę mówiąc jedne otwór mi nie pasował, wiem że przeszkadza wyświetlacz, ale czy nie dało się tego poprzesusować...*

**Hubert Hawłas z Warszawy**

Jako odpowiedź cytuję fragment innego listu **Andrzeja Milewskiego z Torunia:**

*... trafne moim zdaniem było zastosowanie typowego złącza ARK roli gniazda doprowadzającego zasilanie do układu. W wielu konstrukcjach – kitach AVT używacie nietypowych, różnych gniazd zasilających, a każdy elektronik zdaje sobie sprawę, że w tej dobie na rynku nie panuje praktycznie żaden standard, przynajmniej jeżeli chodzi o układy niskonapięciowe...*

Sposób wyprowadzania sygnałów na złącz komputerka opisuje **Krzysztof Wójcik** z Gdańska, mianowicie:

*... zresztą każdy wie, że porządkowanie sygnałów na złączach powoduje zwiększenie komplikacji druku a im więcej przelotek na płytce szczególnie dwustronnej, tym większe prawdopodobieństwo uszkodzenia mechanicznego płytki podczas wielu godzin pracy z komputerem edukacyjnym.... I chwala za to autorowi, bowiem przyznać muszę że wykonując płytki samodzielnie miałem mniej otworów do wiercenia...*

## Też to potrafisz

Sprawa sposobu mocowania obu płytek, obok siebie, czy jedna nad drugą, jest dyskusyjna, w każdym razie otrzymuję od Czytelników zdjęcia komputerków obudowanych w różny sposób. Wszystkie wyglądają wspaniale! Oby tak dalej.

Nieco szerszy list napisał **Przemysław Pierczyk** z Lublina, który postanowił zastosować w czasie świąt komputer do sterowania lampkami na choince – i bardzo dobrze!

... Wykorzystałem do tego specjalne złącze z wyprowadzeniami portu P1 i na próbę zmontowałem układ z 8-miu diod LED tak by każda odpowiadała innemu bitowi portu. Układ próbny miałem gotowy, teraz wystarczył tylko program. ...

Ponieważ Przemek jest ręczniakiem, musiał samodzielnie napisać a następnie zamienić na kod maszynowy programik do sterowania lampkami. Pierwsza wersja programu nie działała poprawnie – powodowała zawieszanie się systemu, druga zaś jak pisze nasz czytelnik działa znakomicie, dlatego przytaczam ja poniżej jako dobry i prosty przykład. Przemek do efektu tzw. bieżącego światła wykorzystał instrukcję rotacji akumulatora (RL A i RR A). Oto listing.

| kod    | etyk  | mnemonik     | komentarz                                                |
|--------|-------|--------------|----------------------------------------------------------|
| 74FE   | pr1:  | mov A, #254  | ;załadowanie do A liczby 254 (11111110)                  |
| 12801A | pr_1: | lcall wait   | ;wywołanie podprogramu ;opóźnienia                       |
| F590   |       | mov 90h, A   | ;wyświetlenie A w porcie P1                              |
| 23     |       | rl A         | ;rotacja bitów A w lewo ;(11111101)                      |
| 309702 |       | jnb 97h, pr2 | ;skok jeśli najstarszy bit P1 jest =0                    |
| 80F5   |       | sjmp pr_1    | ;skok do etykiety pr_1                                   |
| 747F   | pr2:  | mov A, #127  | ;załadowanie do A liczby 127 ;(01111111)                 |
| 12801A | pr_2: | lcall wait   | ;wywołanie programu ;opóźnienia                          |
| F590   |       | mov 90h, A   | ;wyświetlenie A w porcie P1                              |
| 03     |       | rr A         | ;rotacja bitów A w prawo                                 |
| 3090E1 |       | jnb 90h, pr1 | ;skok do pr.1 jeżeli najmłodszy ;bit P1=0 ;następny cykl |
| 80F5   |       | sjmp pr_2    | ;skok do etykiety pr_2 ;(od początku)                    |
|        | wait: |              | ;definicja procedury „wait”                              |
| C0E0   |       | push Acc     | ;przechowaj A na stosie                                  |
| 7433   |       | mov A, #51   | ;czas opóźnienia = 51 x 4 ms ;(ok. 200ms)                |
| 120295 |       | lcall DELAY  | ;wywołanie systemowej ;opóźnienia                        |
| proc.  |       |              |                                                          |
| D0E0   |       | pop Acc      | ;odtworzenie akumulatora                                 |
| 22     |       | ret          | ;i powrót z podprogramu                                  |

Przemysław pyta także o możliwość zastosowania komputerka jako timera- zegara, tematem tym zajmę się w kolejnym odcinku przy okazji omawiania układu przerwań kontrolera i układu licznikowego oraz sposobów praktycznego ich programowania.

I na koniec uwaga Przemka dotycząca treści artykułu klasy mikroprocesorowej z 11 numeru EdV z poprzedniego roku, gdzie objaśniano przykład obliczania skoku do etykiety „w górę”. W artykule wkraść się błąd, napisano mianowicie, że:

... jeżeli policzona ilość bajtów jest równa 10 (0Ah), to przy skoku „w górę” będzie:

U2ofset = 100h – F6h...”

Oczywiście zamiast liczby 100h powinno być FFh, za zwrócenie uwagi na błąd w treści bardzo dziękuję, jak widać chochlik drukarski nie śpi.

\*

**Tomasz Jabłonowski** jako „komputerowiec” w swoim liście porusza sprawę sposobu pisania programów źródłowych tak, aby by-

ły akceptowane przez kompilator oferowany na dyskietce : AVT-2250/D. Otóż aby wszystko było w porządku, jak już wspominałem przy pisaniu programów, obowiązuje zasada, że wszystkie etykiety powinny zaczynać się od pierwszej kolumny tekstu (czyli najbliższej brzości). Pan Tomasz zwraca uwagę, że w artykułach klasy mikroprocesorowej przy opisywaniu kompilatora AVT-2250/D nie podano tego faktu.

Odpowiadam: na dyskietce znajduje się zbiór tekstowy, w którym wymienione są wszystkie zasady pisania programów źródłowych na zawarty kompilator, proszę tylko go przeczytać. W artykułach pominięto niektóre informacje zawarte w treści tego zbioru, ze względu na fakt, że część czytelników tzw. „ręczniaków” nie korzysta w ogóle z kompilatora, i taki opis byłby praktycznie zbędny.

Dlatego zachęcam do przestudiowania zbioru tekstowego zawartego na dyskietce, z pewnością rozwieje on wszystkie wątpliwości.

W swoim jakże ciekawym liście, Tomasz porusza także sprawę prób wyświetlania liczb większych niż te akceptowane przez procedurę A2HEX, opisywaną przy okazji prezentowania Bios’u komputerka. Przytacza prosty programik próbny, dzięki któremu na pozycjach DL3,4, i 5 zostają wyświetlona liczba 123.

Zamieszcza listing tego programu, oto on:

```
include const.inc
include bios.inc

org 8000h
lcall CLS
mov DL3, #_1
mov DL4, #_2
mov DL5, #_3

END
```

Program jest jak najbardziej poprawny, z jednym wyjątkiem, jego wpisanie do komputerka i wykonanie spowoduje że system „pójdzie w maliny”, a to ze względu na brak na końcu programu przed deklaracją END pętli zatrzymującej program w postaci np.:

stop: SJMP stop

Ponieważ mówiłem w artykule dwa miesiące temu o konieczności stosowania tego polecenia, nie będę przypominał konsekwencji wykonania programu, który kończy się bez tej nieskończonej pętli (lub innego skoku bezwzględnego „gdzieś w górę” programu). Proszę zatem pamiętać o stosowaniu tego zalecenia, w przeciwnym przypadku przy wykonaniu tego programu nie zobaczy się efektu swojej pracy, bowiem natychmiast po wyświetleniu liczby 123, na wyświetlaczu prawdopodobnie pojawi się komunikat „-HELLO”, znany wszystkim, a sygnalizujący zresetowanie komputerka.

W kolejnej skrzynce porad omówię pozostałe listy, na razie pozdrawiam wszystkich sympatyków i uczestników Klasy Mikroprocesorowej!

**Sławomir Surowiński**



Dzisiejszy odcinek klasy mikroprocesorowej to pierwsze „po okrągłym roku” spotkanie z Wami. Tak moi drodzy, spotykamy się już tak długo. Jak wynika z listów które otrzymuję od Was, materiał z 12-tu miesięczników, średnio po 10 stron co daje ponad 100 stron materiałów, większości z Was wystarczy aby do tej pory pochwalić się pierwszymi prostymi, aczkolwiek funkcjonalnymi programkami na 8051. Mniej więcej połowa jednak ma nadal pewne problemy, bądź to z pisaniem listingów, bądź ich kompilacją (roczniacy) lub nawet z poprawnym uruchomieniem komputera edukacyjnego który opisałem 9 m-cy temu na łamach EdW. Podejrzewam, że gdybyśmy wszyscy spotykali się w jednej sali (tak jak w szkole) na zajęciach z programowania, wszyscy z pewnością byłiby zadowoleni. Musimy zdać sobie jednak sprawę, że nauka programowania mikrokontrolerów (nie tylko 8051) jest tematem dość złożonym, i wymaga nie tylko cierpliwości, lecz także nieco wysiłku w samodzielnym myśleniu i sięganiu po dodatkową literaturę związaną nie tylko z tematem mikroprocesorów, ale i techniką cyfrową w szczególności.

Namawiam więc wszystkich, was drodzy Czytelnicy do korzystania takiego sposobu nauki, w którym przewodnikiem może być publikowany w EdW cykl, który opracowuję co miesiąc specjalnie dla Was.

A tak na marginesie mam nadzieję, że 13-ty odcinek szkoły mikroprocesorowej nie będzie pechowy, i tym optymistycznym akcentem zapraszam więc do lektury.



W poprzednim odcinku w praktyczny sposób omówiłem port szeregowy mikrokomputera 8051 oraz sposoby transmisji danych poparte prostymi listingami.

Dzisiaj przypomnimy sobie wiadomości na temat omawianych wcześniej układów czasowo-licznikowych oraz układu przerwań procesora. Teraz kiedy zapoznaliście się z językiem procesora, będzie mi łatwiej pokazać w praktycznych przykładach sposób obsługi przerwań i liczników mikrokontrolera.

## Układ przerwań

W jednym z pierwszych odcinków szkoły mikroprocesorowej, przy okazji omawiania końcówek mikroprocesora, podałem przykład – porównanie systemu przerwań mikroprocesora do sytuacji z życia codziennego. Wspominałem że sama nazwa „przerwanie” doskonale odzwierciedla sposób i znaczenie tego układu dla pracy całego mikroprocesora. Jeżeli nie pamiętasz, z czym się „je” temat układu przerwań, radzę przypomnieć sobie ten artykuł. Przejdźmy zatem do konkretów.

W układzie mikrokontrolera 8051 istnieje kilka źródeł przerwań. „Źródła”, czyli dosłownie mówiąc podukładów mikroprocesora, które mogą generować przerwanie. I tak np. weźmy omawiany w poprzednim odcinku port szeregowy. Opisywałem, jak w prosty sposób można np. odebrać znak z portu szeregowego. Pamiętasz, że podałem dwa przykłady. Pierwszy – mniej doskonały polegał na ciągłym sprawdzaniu flagi RI – odbioru znaku, i jeżeli stwierdzaliśmy, że flaga ta została przez procesor ustawiona, to znaczy że odebrano znak, który znajduje się w rejestrze SBUF i jest gotowy do odczytania. Wadą tego sposobu był fakt niekończącego oczekiwania na nadejście bajtu danych z urządzenia zewnętrznego dołączonego do portu szeregowego. Procesor po prostu nie robił nic innego jak tylko czekał na przyjęcie znaku z portu.

Drugi sposób wprowadzał tzw. błąd przeterminowania, kiedy to np. przy braku nadejścia znaku z portu szeregowego, procesor po określonym w programie przez nas czasie przerywał wykonywanie procedury czekania na znak z UART-a, i powracał do programu głównego sygnalizując błąd przeterminowania.

A gdyby tak w pewnych przypadkach dało się uniezależnić odbiór znaków portu szeregowego (szczególnie wtedy gdy nadchodzą one w nieokreślonych przedziałach czasowych) od wykonywania pętli głównego programu?

Otóż tu z pomocą może przyjść system przerwań mikrokontrolera. Ano wyobraźmy sobie sytuację, kiedy program główny wykonuje pewne określone przez nas czynności, natomiast port szeregowy jest ustawiony w taki sposób, że w momencie kiedy zostaje odebrany znak

z UART u, procesor automatycznie przerywa wykonywanie programu i wykonuje skok do tzw. „procedury (podprogramu) obsługi przerwania z portu szeregowego”. Po wykonaniu tej procedury – napisanej oczywiście przez programistę i kończącej się instrukcją

### RETI

program powraca do kolejnej instrukcji pętli głównej która następuje po tej przy której nastąpiło przerwanie.

A co się dzieje w podprogramie obsługi przerwania? Co nam przyjdzie do głowy, czyli przede wszystkim przepisanie danej z rejestru SBUF do innego rejestru, z obszaru wewnętrznej pamięci danych procesora, aby nie stracić cennego znaku, kiedy przyjdzie następny (i SBUF zostanie ponownie zmieniony).

To tylko ilustracja wykorzystania systemu przerwań do wspomnienia pracy programu mikroprocesora.

Przejdźmy jednak od szczegółowego omówienia źródeł wszystkich przerwań w mikrokontrolerze 8051 i sposobów ich wykorzystania w naszych programach.

Układ przerwań procesora może przyjmować zgłoszenia następujących przerwań:

- zewnętrzne: z wejść /INT0 i /INT1 (piny P3.2 – 12, P3.3 – 13) (2 przerwania)
- z portu szeregowego (jedno przerwanie)
- z układu licznikowego: przepełnienie licznika T0, lub T1 oraz w przypadku procesora 80C52 – z układu licznikowego T2 (dwa przerwania dla 8051)

Zanim przejdziemy do omówienia każdego ze źródeł przerwań powinniśmy sobie uzmysłowić fakt istnienia tzw. znaczników każdego z przerwań. Znacznik fizycznie jest pojedynczym bitem zawartym w kilku rejestrach SFR procesora. W przypadku początkowym znacznik danego przerwania jest wyzerowany – do bitu wpisane jest zero. W przypadku ustawienia znacznika przerwania (wpisania do niego jedynki) następuje zgłoszenie przerwania. Wyzerowania znacznika to anulowanie zgłoszenia. Każde z wymienionych przerwań posiada własny znacznik zgłoszenia przerwania. Znaczniki przerwań są ustawiane automatycznie przez procesor w momencie wystąpienia warunku nadejścia przerwania, i tak dla poszczególnych źródeł będą to:

- /INT0, /INT1 : opadające zbocze sygnału na tym wejściu (lub poziom niski)
- zakończenie odbioru lub nadawania znaku przez UART
- przepełnienie licznika T0 lub T1 (zmiana zawartości z 0FFF na 0000h)

## Też to potrafisz

Ponieważ jednocześnie w programie może pracować kilka układów generujących przerwania, a same przerwania czasem nie są nam potrzebne, istnieje specjalny rejestr w obszarze SFR o nazwie IE (ang. „Interrupt Enable” – zezwolenie na przerwanie), dzięki któremu możemy uaktywniać lub blokować generowanie wybranych przerwań. Poszczególne bity tego rejestru odpowiadają za generowanie przerwania od określonego podbloku mikrokontrolera, a dodatkowo najstarszy bit tego rejestru pozwala na bezwarunkowe wyłączenie systemu przerwań. Bity te często nazywa się maskującymi, co w praktyce oznacza, że wpisanie do niego jedynki powoduje uaktywnienie danej funkcji bitu, wyzerowanie zaś – to zablokowanie.

Oto rejestr IE (adres w SFR A8h) i znaczenie jego poszczególnych bitów:

| bity: | AFh | A Eh | ADh | AC h | ABh | AAh | A9h | A8h |    |
|-------|-----|------|-----|------|-----|-----|-----|-----|----|
| A8h   | EA  | -    | ET2 | ES   | ET1 | EX1 | ETO | EXO | IE |
|       | 7   | 6    | 5   | 4    | 3   | 2   | 1   | 0   |    |

**EA (bit IE.7, adres: AFh)** – bit aktywacyjny systemu przerwań (=0 wszystkie przerwania są zablokowane, =1 odblokowane są te przerwania, których bit jest ustawiony)

**bit IE.6** o adresie AEh jest nie wykorzystany

**ET2 (bit IE.5, adres: ADh)** – tylko w 80C52 (8052) bit maskujący przerwanie z licznika T2

**ES (bit IE.4, adres: ACh)** – bit maskujący przerwanie z portu szeregowego

**ET1 (bit IE.3, adres: ABh)** – bit maskujący przerwanie z licznika T1

**EX1 (bit IE.2, adres: AAh)** – bit maskujący przerwanie z wejścia /INT1

**ETO (bit IE.1, adres: A9h)** – bit maskujący przerwanie z licznika T0

**EXO (bit IE.0, adres: A8h)** – bit maskujący przerwanie z wejścia /INT0

Czyli jeżeli np. chcemy uaktywnić przerwanie z wejścia zewnętrznego INT1, należy wykonać instrukcje:

```
SETB EX1 ;uaktywnienie przerwania z INT1
SETB EA ;globalne odblokowanie przerwań
```

W przypadku chęci uaktywnienia kilku przerwań np. z licznika T0 i układu transmisji szeregowego UART, można wykonać następujące operacje:

```
SETB ETO ;uaktywnienie przerwania od T0
SETB ES ; uaktywnienie przerwania od UART a
SETB EA ;globalne odblokowanie przerwań
```

Ponieważ wszystkie bity maskujące przerwania znajdują się w jednym rejestrze, można w/w instrukcję zapisać jako jedną:

```
MOV IE, #10010010b
```

prawda, że proste. W praktyce jeżeli chcemy np. na jakiś czas wyłączyć jakieś przerwanie np. od licznika T0 wystarczy wykonać instrukcję:

```
CLR ETO ;zablokowanie przerwania od T0
```

Jeżeli zaś zachodzi potrzeba zablokowania całego systemu przerwań można tego dokonać wyzerowując bit EA za pomocą instrukcji :

```
CLR EA
```

lub

```
ANL IE, #7Fh ;7Fh = 01111111b
```

Ten pierwszy sposób, kiedy operujemy na bitach jest bardziej czytelny, dlatego polecam go w praktyce.

Ważną informacją jest fakt, że po resecie procesora rejestr IE jest wyzerowany, co oznacza że wszystkie przerwania są zablokowane (EA=0) oraz dodatkowo maski wszystkich przerwań są także zablokowane (IE.5 – IE.0 = 0).

Ze względu na fakt że przerwania nie są przeważnie generowane rozmyślnie poprzez instrukcje programisty (a jak?... to proste przecież poprzez ustawienie jednego ze znaczników przerwań – o tym za chwilę) ale przez podbloki wykonawcze procesora, zatem może się zdarzyć sytuacja, kiedy to w jednej chwili nadejdą dwa lub więcej przerwań – np. w tej samej chwili na wejściu INT1 pojawi się stan niski (generując przerwanie od /INT1) oraz przepelni się licznik T0 (generując przerwanie od T0), i co wtedy, czy nie nastąpi zatem jakiś konflikt? Otóż nie. Okazuje się bowiem, że projektanci mikrokontrolerów 8051 i pochodnych pomyśleli o takiej sytuacji i ustalili, że każde przerwanie procesor posiada odpowiedni **priorytet**. To bardzo ważne słowo i dlatego warto je dobrze zapamiętać.

**Priorytet** danego przerwania nad innym, w praktyce to znaczy, że w przypadku kiedy zajdzie przypadek jak przedstawiony powyżej, kiedy w jednej chwili zachodzą dwa różne przerwania, to w pierwszej kolejności zostanie przyjęte przerwanie o wyższym priorytecie i wykonana zostanie stosowna dla niego procedura obsługi przerwania (czyli nic innego jak kawałek napisanego przez ciebie programu zakończony instrukcją RETI). Przerwanie drugie – o niższym priorytecie będzie jak gdyby „czekać na swoją kolej”, a kiedy ta przyjdzie, zostanie ono przyjęte i wykonana zostanie procedura obsługi tegoż drugiego przerwania (także zakończona instrukcją RETI).

I tak ze względu na to że mamy w procesorze 8051 pięć (w 8052 sześć) źródeł przerwań, każde z nich posiada odpowiedni priorytet, i tak w kolejności od najmniejszego priorytetu do największego jest:

ET2, ES, ET1, EX1, ET0, EX0, czyli

najmniej uprzywilejowanym jest przerwanie od licznika T2 (w 8052), dalej w kolejności (jak w rejestrze IE) większy priorytet ma przerwanie z portu szeregowego UART, dalej od licznika T1, z wejścia INT1, wreszcie od licznika T0, a najbardziej uprzywilejowanym jest przerwanie z wejścia /INT0.

Ktoś zapyta, a co się stanie, jeżeli nadchodzi przerwanie np. z wejścia INT1 i rozpoczęte zostaje wykonywanie procedury obsługi przerwania dla tego wejścia, a w czasie jej trwania nadchodzi przerwanie o wyższym priorytecie np. z wejścia INT0? A no wtedy procedura obsługi przerwania z INT1 zostaje natychmiast przerwana i procesor skacze do procedury obsługi przerwania z wejścia INT0. Kiedy ją skończy, powraca (skacze) do miejsca skąd nastąpił skok gdy nadeszło przerwanie o wyższym priorytecie i program toczy się dalej, prawda że logiczne rozwiązanie. Jak nad tym wszystkim zapanować tak, aby program nie „poszedł w maliny”, opowiem za chwilę.

„...No dobrze, już wiem, o co chodzi z tym priorytetem przerwań, ale przecież może zajść przypadek, kiedy mam taki układ elektroniczny, w którym zastosowany procesor musi wykorzystywać przerwania z nieco inną kolejnością priorytetów, i co wtedy? Czy jestem skazany na ustaloną kolejność priorytetów przerwań? Odpowiedź brzmi nie. Otóż istnieje dodatkowy specjalny rejestr tzw. priorytetów przerwań o nazwie IP (ang. „Interrupt Priority” – priorytet przerwania). Znajduje się on pod adresem B8h jak się zapewne domyślasz w obszarze SFR procesora. Dzięki niemu można zmieniać priorytety poszczególnych przerwań wymienionych wcześniej, powodując że dane przerwanie mające dotąd niższy priorytet może uzyskać wyższy, dzięki odpowiedniemu ustawieniu bitów w rejestrze priorytetu IP. Oto szczegółowe znaczenie poszczególnych bitów tego rejestru:

| bity: | BFh | BEh | BDh | BC h | BBh | BAh | B9h | B8h |    |
|-------|-----|-----|-----|------|-----|-----|-----|-----|----|
| B8h   | -   | -   | PT2 | PS   | PT1 | PX1 | PT0 | PX0 | IP |
|       | 7   | 6   | 5   | 4    | 3   | 2   | 1   | 0   |    |

**bity IP.7 i IP.6** – nie wykorzystane

**PT2 (bit IP.5, adres: BDh)** – bit priorytetu przerwania z licznika T2 (tylko w 80C52, 8052)

**PS (bit IP.4, adres: BCh)** – bit priorytetu przerwania z portu szeregowego.

**PT1 (bit IP.3, adres: BBh)** – bit priorytetu przerwania z licznika T1

**PX1 (bit IP.2, adres: BAh)** – bit priorytetu przerwania z wejścia /INT1

**PT1 (bit IP.3, adres: BBh)** – bit priorytetu przerwania z licznika T1

**PX1 (bit IP.2, adres: BAh)** – bit priorytetu przerwania z wejścia /INT1

**PT0 (bit IP.1, adres: B9h)** – bit priorytetu przerwania z licznika T0

**PX0 (bit IP.0, adres: B8h)** – bit priorytetu przerwania z wejścia /INT0

I tak ustawienie jednego z bitów powoduje ustawienie danego przerwania na wyższym poziomie i odwrotnie, wyzerowanie danego bitu powoduje ustawienie niższego priorytetu danego przerwania. W przypadku kiedy ustawimy wyższy priorytet kilku przerwań na raz, o kolejności wykonywania poszczególnych procedur obsługi przerwań decyduje ustalona wcześniej kolejność dla przypadku kiedy wszystkie bity rejestru IP są równe 0. Warto zatem powiedzieć sobie, że podprogram (procedura) obsługi przerwania z umieszczonego na najwyższym poziomie jest nieprzerwywalna. W przypadku np. kiedy IP=0, będzie to przerwanie z wejścia /INT0.

Po resecie procesora podobnie jak w przypadku rejestru IE, wszystkie bity rejestru IP są wyzerowane (IP=0).

„...No dobrze ale co fizycznie zachodzi, kiedy nadchodzi przerwanie, i o co chodzi z tą procedurą obsługi przerwania? Oto wyjaśnienie.

Otóż kiedy zajdzie warunek przerwania (kiedy znacznik danego przerwania zostaje ustawiony), np. kiedy nadejdzie zbocze opadające na wejściu /INT1, przy ustawionym bicie EX1 oraz uaktywnionym systemie przerwań (EA=1) procesor wykona następujące operacje

- po pierwsze: sprawdzi czy nie jest wykonywana akurat procedura obsługi przerwania o wyższym priorytecie lub czy jednocześnie nie nadeszło przerwanie o wyższym priorytecie z innego źródła
- po drugie (jeżeli nie zdarzy się warunek z pierwszego): wyzeruje znacznik zgłoszenia przyjętego przerwania. I tu wyjątek, nie są bowiem automatycznie zerowane znaczniki przerwań: z portu szeregowego T1 – przy wysłaniu znaku, RI – przy nadejściu znaku, oraz z licznika T2 w przypadku procesora 8052/C52.
- po trzecie: procesor zapisze na stosie zawartość 16-bitowego licznika rozkazów PC
- i po czwarte : procesor automatycznie wpisze do licznika rozkazów PC ustalony fabrycznie adres początku programu (procedury) obsługi danego przerwania, oto te adresy dla poszczególnych przerwań:

0003h – dla przerwania z wejścia /INT0  
 000Bh – dla przerwania z licznika T0  
 0013h – dla przerwania z wejścia /INT1  
 001Bh – dla przerwania z licznika T1  
 0023h – dla przerwania z portu szeregowego  
 oraz dodatkowo w procesorach 8052/C52  
 002Bh – dla przerwania z licznika T2

Jak zauważyliście, przytoczone adresy są ustalone fabrycznie a oddalone od siebie dokładnie o 4 bajty. Dlaczego akurat o cztery, zaraz się okaże. W każdym razie zanim to wyjaśnię, spróbuję oswoić Was przy tej okazji z określeniem takiej struktury adresów, które obowiązują nie tylko w nazewnictwie związanym z 8051, ale wszystkimi układami mikroprocesorowymi, a mianowicie nazwą: **tablice wektorów przerwań**.

**Tablice** – bo poszczególne adresy oddalone dodatkowo równo od siebie (o 4 bajty) mogą kojarzyć się z tablicą

**Wektorów** – bo ze względu na tylko 4 bajty przeznaczone na podprogram, fizycznie nie zapiszemy w tym miejscu podprogramu, a jedynie wypiszemy wskaźnik (wektor) pokazujący gdzie w programie (pod jakim adresem) znajduje się właściwa procedura obsługi danego przerwania.

**Przerwań** – bo oczywiście cały zwrot dotyczy przerwań.

„...Co oznaczają te adresy, i jak je wykorzystać?” Otóż jak zapewne pamiętacie po resecie procesora licznik rozkazów jest wyzerowany, co oznacza że procesor rozpoczyna wykonywanie programu od adresu 0000h.

Z drugiej strony zauważcie, że pomiędzy adresem 0000h a adresami procedur obsługi przerwań znajdują się zawsze 4 bajty na... program, czy to aby nie za mało? Otóż nie!

Istnieje przecież w liście rozkazów mikrokontrolera 8051 instrukcja skoku bezwzględnego pod wskazany adres. Jest to LJMP, czasem AJMP (SJMP)

Aby zbyt nie namieszać Wam w głowach posłużę się przykładem.

Załóżmy że chcemy napisać program, w którym wykorzystamy dwa źródła przerwań: pierwsze z wejścia /INT0 drugie z licznika T1.

Generalnie zatem program będzie składał się z:

- instrukcji pętli głównej programu
- oraz dwóch procedur (podprogramów) obsługi przerwań: pierwsza dla wejścia /INT0, druga dla licznika T1. Podprogramy z reguły są ciągiem minimum kilku instrukcji, które przecież nie zmieszczą się w 4 bajtach! Użyjemy więc wektorów „przekierowujących” program z tablicy wektorów przerwań do właściwego miejsca w programie gdzie znajduje się właściwy dla danego przerwania podprogram.

Przykładowy listing takiego programu mógłby wyglądać następująco:

```
;początek programu
ORG 0000h ;początek wykonywania programu
LJMP START ;skocz do etykiety początku programu
 ;głównego
```

```
;tablica wektorów przerwań
ORG 0003h ;pod adresem 0003h umieszczam
LJMP intEX0 ;wektor – czyli instrukcję skoku do
 ;procedury intEX0
ORG 001Bh ;a pod adresem 001Bh umieszczam
LJMP intT1 ;wektor do procedury obsługi
 ;przerwania od T1
```

;właściwy początek pętli głównej programu

```
START:
..... ;instrukcje inicjujące (początkowe)
SETB EX0 ;uaktywnienie przerwania z /INT0
SETB ET1 ;uaktywnienie przerwania z T1
SETB EA ;globalne odblokowanie przerwań
..... ;inne instrukcje pętli głównej
.....
```

```
;tu początek podprogramu obsługi przerwania z /INT0
intEX0:
```

```
..... ;instrukcje dotyczące
..... ;procedury w przypadku
..... ;nadejścia przerwania z /INT0
reti
```

```
intT1:
..... ;instrukcje dotyczące
..... ;procedury w przypadku
..... ;nadejścia przerwania z T1
reti
```

```
END ;koniec programu
```

Tak więc jak widać z przytoczonego listingu w tablicy wektorów przerwań umieszczone zostały jedynie skoki bezwzględne dla każdego

z przerwań do miejsc w programie gdzie rozpoczynają się właściwe procedury ich obsługi.

W praktyce w zależności od rozmiarów kodu programu można użyć także instrukcji AJMP ale tylko kiedy wszystkie podprogramy znajdują się w pierwszych 2 kilobajtach kodu programu (czyli o adresach: 0000h...07FFh). Można także umieszczać podprogramy obsługi przerwań w różnych miejscach pamięci programu w stosunku do pętli głównej np. przed nią (w naszym przykładzie przed etykietą START). Czasami, w przypadku kiedy rozmiar dostępnej pamięci programu jest dość krytyczny – (brakuje nam pamięci) aby nie marnować drogocennych bajtów, można także zrezygnować ze skoku typu LJMP, i rozpocząć procedurę obsługi przerwania od adresu z tabeli wektorów przerwań. Oczywiście dotyczy to przypadku, kiedy wspomniane przerwanie albo jest jedyne w uaktywnionych w systemie, albo jest na ostatnim miejscu w tabeli wektorów przerwań. W przypadku naszego przykładu listing mógłby wyglądać następująco:

```
;początek programu
ORG 0000h ;początek wykonywania programu
LJMP START ;skocz do etykiety początku programu
 ;głównego
```

```
;tablica wektorów przerwań
ORG 0003h ;pod adresem 0003h umieszczam
LJMP intEX0 ;wektor – czyli instrukcję skoku do
 ;procedury intEX0
ORG 001Bh ;a tu zaczyna się procedura obsługi
 ;przer. od T1
```

```
intT1:
..... ;instrukcje dotyczące
..... ;procedury w przypadku
..... ;nadejścia przerwania z T1
reti
```

```
;właściwy początek pętli głównej programu
START:
```

```
..... ;instrukcje inicjujące (początkowe)
SETB EX0 ;uaktywnienie przerwania z /INT0
SETB ET1 ;uaktywnienie przerwania z T1
SETB EA ;globalne odblokowanie przerwań
..... ;inne instrukcje pętli głównej
.....
```

```
;tu początek podprogramu obsługi przerwania z /INT0
intEX0:
```

```
..... ;instrukcje dotyczące
..... ;procedury w przypadku
..... ;nadejścia przerwania z /INT0
reti
END ;koniec programu
```

Oczywiście w zasadzie etykieta intT1 jest w tym przypadku zbędna, informuje jedynie o tym że w tym miejscu zaczyna się procedura obsługi przerwania od licznika T1.

Można by oczywiście przetrzucić procedurę intEX0 w miejsce pomiędzy instrukcją kończącą procedurę intT1 a etykietę START, w praktyce najczęściej nie ma to żadnego znaczenia.

Program obsługi przerwania musi być zakończony instrukcją RETI. Do tej instrukcji nie zostanie przyjęte zgłoszenie żadnego przerwania z poziomu równego (tego samego) lub niższego. Wreszcie kiedy procesor wykona instrukcję kończącą podprogram przerwania (RETI) odtwarzany jest ze stosu adres powrotu sprzed wywołania i wpisany do licznika rozkazów PC procesora, po czym program główny toczy się dalej.

Jeżeli chodzi o znaczniki zgłoszenia przerwań to można je „wylłowić” z wkładki z EdV nr 11/97, gdzie znajduje się skrócony opis wszystkich rejestrów SFR procesora. Oto one:

- z wejścia /INT0: bit IT0 (adres: 88h) w rejestrze TCON (88h)
- z wejścia /INT1: bit IT1 (adres: 8Ah) w rejestrze TCON (88h)
- z licznika T0: bit TF0 (adres: 8Dh) w rejestrze TCON (88h)
- z licznika T1: bit TF1 (adres: 8Fh) w rejestrze TCON (88h)
- z portu szeregowego: bit TI w przypadku nadania znaku (adres: 99h) oraz bit RI w przypadku odbioru znaku z portu (adres: 98h) z rejestru SCON (98h)
- z licznika T2: bit TF2 (adres: CFh) – dla przepełnienia licznika T2 oraz bit EXF2 (adres: CEh) – przy wykryciu opadającego zbocza sygnału na tym wejściu (P1.1 w 8052/C52), oba znaczniki z rejestru sterującego pracą licznika T2 – T2CON (C8h).

Ponieważ sprawą układu czasowo-licznikowego zajmę się w drugiej części artykułu, pozostaje mi do omówienia kilka dodatkowych informacji dotyczących obsługi i generowania przerwań zewnętrznych z wejść /INT0 i /INT1.



## Też to potrafisz

I tak na wstępie ważna informacja: przerwania te mogą być zgłaszane opadającym zboczem sygnału na tym wejściu (zmiana z poziomu logicznego H na L) lub poziomem niskim sygnału. Wybór należy do programisty i może być zmieniany za pomocą odpowiedniego rejestru, ale o tym za chwilę.

W praktyce różnica pomiędzy tymi dwoma typami zgłaszania przerwania polega na tym, że:

- w przypadku zgłoszenia przerwania opadającym zboczem: procedura zostanie wywołana tylko jeden raz, nawet jeżeli pierwsze jej wywołanie zostanie zakończone (instrukcja RETI) a stan na wejściu /INTx po jej zakończeniu nadal jest niski.
- w przypadku zgłoszenia przerwania poziomem niskim: poziom na wejściu /INTx powinien zmienić się znowu na wysoki przed zakończeniem procedury obsługi przerwania (przed instrukcją RETI) w przeciwnym przypadku procedura obsługi zostanie wywołana ponownie.

A oto wspomniany rejestr kontrolujący przerwania zewnętrzne: TCON (adres: 88h). Starsze 4 bity z tego rejestru obsługują układy czasowo licznikowe, toteż omówieniem ich zajmę się za chwilę.

|            |     |     |     |     |     |     |     |     |      |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|------|
| bity:      | 8Fh | 8Eh | 8Dh | 8Ch | 88h | 8Ah | 89h | 88h |      |
| <b>88h</b> | TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 | TCON |
|            | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |      |

Bity dotyczące wejść /INT0 i /INT1 procesora:

**IE1 (bit TCON.3, adres: 8Bh)** – znacznik zgłoszenia przerwania na wejściu /INT1. Jest ustawiany sprzętowo po wykryciu zgłoszenia. Zerowany automatycznie przy przyjęciu przerwania (przy wejściu do procedury obsługi)

**IT1 (bit TCON.2, adres: 8Ah)** – bit sterujący sposobem zgłoszenia przerwania na wejściu /INT1: opadającym zboczem (IT1=1) lub poziomem niskim (IT1=0) sygnału zewnętrznego

**IE0 (bit TCON.1, adres: 89h)** – znacznik zgłoszenia przerwania na wejściu /INT0. Jest ustawiany sprzętowo po wykryciu zgłoszenia. Zerowany automatycznie przy przyjęciu przerwania (przy wejściu do procedury obsługi)

**IT0 (bit TCON.0, adres: 88h)** – bit sterujący sposobem zgłoszenia przerwania na wejściu /INT0: opadającym zboczem (IT0=1) lub poziomem niskim (IT0=0) sygnału zewnętrznego

Analizując sposób zgłaszania przerwania zewnętrznych nie sposób nie powiedzieć w jaki sposób fizycznie procesor rejestruje zajście zgłoszenia przerwania. Czy np. wejścia /INTx mają na wejściu jakieś przerzutniki? Otóż nie. Procesor w pewnych okresach każdego cyklu maszynowego próbuje stan wejść /INTx, i jeżeli w dwóch kolejnych cyklach stwierdzi zmianę stanu z 1 na 0 oznaczało to będzie że nastąpił warunek zgłoszenia przerwania. Dokładne zależności czasowe pomiędzy fizyczną zmianą poziomu na wejściu przerywającym /INTx a zgłoszeniem przerwania można znaleźć w katalogach procesorów 8051 różnych producentów (Philips, Atmel, itp.)

Ponieważ w praktyce rzadko zachodzi potrzeba takiej analizy, nadmienię, żeby wobec braku sprzętowych „przerzutników rejestrujących” opadające zbocza na wejściach /INTx, każdy z sygnałów przerywających (generowanych na końcówkach /INTx) trwał co najmniej przez 12 taktów zegarowych procesora.

W praktyce oznacza to, że np. w przypadku procesora pracującego z kwarcem 12MHz najkrótsza jedynka i zero generowana na tym wejściu (przez układ zewnętrzny) wystarczająca jednakże do wywołania przerwania w programie procesora, powinna trwać nie mniej niż 1 us (mikrosekundę – każdy poziom)

W sumie wyszło nam, że można już łapać ujemne impulsy o czasie trwania 1 us (poziom niski).

Z pewnością niektórzy z czytelników odwrócą sytuację i stwierdzą, że przerwanie /INTx można teoretycznie generować przy takim kwarcu prawie 500 000 razy na sekundę (500kHz), tylko po co?

Zresztą gdyby ktoś np. uaktywnił przerwanie np. /INT0 i do tego wejścia /INT0 dołączył generator przebiegu TTL o takiej częstotliwości, to program procesora w praktyce „zawariowałby”, bowiem co chwilę wywoływana byłaby procedura obsługi przerwania z INT0, i nic innego w programie nie byłoby wykonywane. Dlatego pamiętając o tym należy rozsądnie wybierać zastosowania układu przerwań zewnętrznych pamiętając także o występujących tu ograniczeniach.

Na koniec omawiania układu przerwań nie można zapomnieć o praktycznej wskazówce dotyczącej pisania procedur obsługi przerwania. I tak przypomnijmy sobie stwierdzenie, mówiące że procedura obsługi przerwania rozpoczyna się w chwili nadejścia przerwania. Ponieważ generowanie przerwania zazwyczaj zależy od mniej lub bardziej złożonych czynników zewnętrznych, i nie jest z reguły przewidywalne w programie, może nastąpić sytuacja tzw. gubienia zawartości rejestrów roboczych (np. akumulatora) po wykonaniu procedury obsługi przerwania.

Wyobraźmy sobie sytuację, kiedy to procesor spokojnie i „sielankowo” wykonuje napisany przez siebie program główny i np. w tej chwili próbuje dodać dwie liczby znajdujące się w rejestrach A i B, po czym będzie chciał wypisać je na wyświetlaczu naszego komputerka edukacyjnego korzystając ze standardowej procedury A2HEX (dla uproszczenia przyjmujemy wypisanie wyniku bez najstarszego bitu wyniku umieszczonego w C).

Aby to wykonać zapewne posłuży się instrukcjami:

```
MOV A, #skladnik1 ;załadowanie składowika (1)
ADD A, #skladnik2 ;i dodanie składowika 2 (2)
MOV B, #1 ;na 1-szej pozycji (3)
LCALL A2HEX ;wypisz zawartość Acc (4)
..... ;i rób coś dalej (5)
```

W nawiasach podano numery linii.

Popatrzmy, niech no wykonane zostaną instrukcje z linii (1) i (2) oraz (3), w tym momencie, przed wykonaniem linii (4), kiedy wypisany zostaje wynik, następuje zgłoszenie jakiegoś przerwania (obojętne jakiego) o program automatycznie skacze do odpowiedniego wskaźnika z tablicy wektorów przerwań, a z tamtą prawdopodobnie w innym miejscu programu, gdzie znajduje się właściwa dla danego zdarzenia procedura obsługi przerwania.

Dla przykładu powiedzmy, że procedura ta wykonuje pewne obliczenia i operacje na rejestrach, w tym m.in. na akumulatorze, np.

```
intT2:
MOV A, LICZNIK
ADD A, #1
DA A
MOV LICZNIK, A
RETI
```

No dobrze, po zgłoszeniu przerwania i wykonaniu instrukcji zawartych w procedurze procesor po wykonaniu instrukcji RETI powróci do linii (4) programu głównego i... no właśnie, wypisana zostanie nie wartość sumy dwóch składowików, ale zawartość jakiegoś rejestru LICZNIK (zdefiniowanego gdzieś wcześniej w programie przez programistę). W efekcie wyświetlacz pokaże bzdury, a my nie będziemy wiedzieli co się stało.

Takich sytuacji może być bardzo wiele. Jak zatem w prosty sposób można się zabezpieczyć przed skutkami modyfikacji rejestrów podczas wykonywania pojawiających się często „nie stąd ni z owąd” procedur obsługujących przerwanie? Metoda jest bardzo prosta i polega na zapamiętywaniu wartości używanych w danej procedurze rejestrów na początku tej procedury, po czym przed końcem procedury obsługi przerwania – odtworzenie ich pierwotnej zawartości i wykonaniu standardowej już instrukcji powrotu z przerwania RETI.

Najprostszym i zdecydowanie polecanym, a praktycznie jedynym sensownym sposobem zapamiętywania i odtwarzania rejestrów jest korzystanie ze stosu.

Oto poprzedni przykład zmodyfikowany w sposób zabezpieczający zawartość akumulatora przed przypadkową utratą bieżącej „wartości”, intT2:

```
PUSH Acc ;zapamiętanie akumulatora
MOV A, LICZNIK
ADD A, #1
DA A
MOV LICZNIK, A
POP Acc ;odtworzenie akumulatora
RETI
```

Jak widać stos w tym przypadku oddał nam niesamowitą przysługę, bowiem za pomocą jednej instrukcji odłożenia na stos a następnie zdjęcia nie zakłóciliśmy toku wykonywania części głównej programu – akumulator pozostał ten sam, a wynik na wyświetlaczu będzie z pewnością poprawny.

Ktoś może w tym miejscu powiedzieć, że przecież można by podzielić używane rejestry (w końcu w procesorze jest ich dużo...) na dwie grupy w tym przypadku, jedna byłaby modyfikowana w programie głównym, a druga wykorzystywana by była tylko w procedurze obsługi przerwania. Gdyby dało się tak zrobić w praktyce, byłoby wspaniale, rzeczywistość jest jednak nieco mniej różowa. Co zrobić bowiem z rejestrami uniwersalnymi np. jednostki arytmetyczno – logicznej ALU? Przecież jest tylko jeden rejestr Acc oraz np. rejestr B (nie mówiąc o wskaźniku danych DPTR).

Odpowiedź jest jedna: używać stosu.

W przypadku gdy w procedurze obsługi przerwania modyfikowanych jest więcej niż jeden rejestr, należy „odkładać je” i „zdejmować” ze stosu w sposób zgodny z zasadą działania samego stosu, a mianowicie, „to co pierwsze odłożyliśmy to ostatnie zdejmujemy”, czyli np. jeżeli w naszym przykładzie procedury intT2 zaprzęgniemy dodatkowy rejestr, prawidłowa kolejność instrukcji będzie następująca.



intT2:

```

PUSH Acc ;zapamiętanie akumulatora
PUSH B ;zapamiętanie rejestru B
MOV A, LICZNIK1
ADD A, #1
DA A
MOV B, #3
MUL AB
MOV LICZNIK1, A
MOV LICZNIK2, B
POP B ;odtworzenie rejestru B
POP Acc ;odtworzenie akumulatora
RETI

```

Przy okazji omawiania systemu przerwań nie sposób nie wspomnieć o dwóch głównych, często popełnianych błędach początkujący programistów podczas pisania pierwszych programów wyposażonych w procedur obsługi jednego lub kilku przerwań.

**Pierwszy błąd** polega na wykorzystywaniu zbyt dużej liczby rejestrów w procedurze obsługi przerwań, a co za tym idzie konieczności odkładania na stos zbyt dużej liczby danych. W efekcie często (szczególnie w przypadkach kiedy pracują więcej niż jeden źródła przerwań) stos zostaje przepełniony – tzn. że wskaźnik stosu zostaje zwiększony do wartości pod którą w pamięci wewnętrznej RAM programista zaplanował mniej lub bardziej ważne (ale ważne i przewidziane w programie!) zmienne programowe. W takim przypadku zmienne te zostaną z pewnością zamazane, i nasz program będzie do niczego. O tym jakie są sposoby omijania takich przypadków, opowiem za chwilę.

**Drugi błąd** polega na tym że programista tworzy „zbyt długi” kod procedury obsługi przerwań. Przecież każda instrukcja zajmuje procesorowi określoną ilość czasu! W efekcie np. w sytuacji kiedy przerwanie zewnętrzne (lub z przepełnienia licznika) nadchodzi odpowiednio często – czyli w określonych przedziałach czasu, może dojść do sytuacji, kiedy to w trakcie trwania nie zakończonej jeszcze procedury obsługi przerwań znajdzie ponowny warunek zgłoszenia przerwań. W większości takich przypadków procesor po prostu „zwaruje” a cały program albo się zawiesi, albo pójdzie w przysłowiowe „maliny”.

Unikajmy więc takich przypadków, i piszmy procedury obsługi przerwań w taki sposób aby nie powodować krytycznych błędów czasowych, a przynajmniej zabezpieczajmy się przed nimi.

Oczywiście nie w każdym przypadku obowiązuje zasada pisania krótkich procedur obsługi przerwań. Bywają przypadki (z doświadczenia powiem Wam że należy do nich kompleksowa procedura obsługi sygnału DCF77 i dekodowanie aktualnych danych o dacie i czasie), kiedy procedura jest na pierwszy rzut oka dość długa. Lecz sposób jej działania oraz maksymalny czas niezależnie od cyklu, jest przemyślany w taki sposób, aby pozostawić bezpieczny margines czasowy i co najważniejsze dać czas procesorowi także na wykonywanie procedur obsługi innych przerwań, a co najważniejsze, na wykonanie instrukcji części głównej programu. Wszystko to w warunkach kiedy mamy do czynienia z małymi wartościami częstotliwości pracy procesora przy dość często zachodzących przerwanach tak zewnętrznych jak i wewnętrznych.

Na koniec omawiania systemu przerwań warto wspomnieć o przewidzianym w zasadzie do obsługi podprogramów przerwań systemie bloków rejestrów roboczych: R0, R1, R2...R7. I tak w przestrzeni adresowej wewnętrznej RAM procesora (adresy 0...127) w pierwszych 32 rejestrach (adresy: 0...31) przewidziano cztery „banki” rejestrów R0...R7. Dostęp do nich za pomocą operowania instrukcjami wykorzystującymi nazwy rejestrów roboczych R0...R7, jest możliwy za pomocą odpowiedniego ustawienia dwóch bitów w omawianym już w naszym cyklu słowie PSW (ang. „Program Status Word”, SFR adres: D0h). Są to bity RS0 (adres: D3h) i RS1 (adres: D4h). I tak w zależności od kombinacji tych bitów uzyskujemy dostęp poprzez nazwy robocze R0...R7 do następujących rejestrów w wewn. RAM procesora:

| RS1 | RS0 | bank | adresy fizyczne R0...R7 w wewn. RAM |
|-----|-----|------|-------------------------------------|
| 0   | 0   | 0    | 00h - 07 (0...7)                    |
| 0   | 1   | 1    | 08h - 0Fh (8...15)                  |
| 1   | 0   | 2    | 10h - 17h (16...23)                 |
| 1   | 1   | 3    | 18h - 1Fh (24...31)                 |

W przypadku korzystania z tej cechy adresowania rejestrów roboczych procesora, należy pamiętać o odpowiednim przesunięciu wskaźnika stosu (który na początku po resetie procesora zawsze wskazuje na adres 07h) w zależności od ilości wykorzystywanych banków rejestrów R0...R7, która to ilość często wiąże się z ilością używanych przerwań

w systemie. W przypadku używania np. wszystkich czterech banków oraz dodatkowo korzystania ze stosu (choć w ograniczonym zakresie) w procedurach obsługi przerwań, na początku programu należy wykonać instrukcję:

```
MOV SP, #1Fh ;przesunięcie wskaźnika stosu
```

co spowoduje że żaden z 32 rejestrów roboczych (4 banki po 8 – R0...R7) nie zostanie zamazany w przypadku odłożenia jakiejś zmiennej w procedurze obsługi przerwań.

Oczywiście używanie pojęcia banków oraz takiej architektury rejestrów roboczych nie jest wymagane, można przecież adresować każdy z nich (32) bezpośrednio za pomocą odpowiednich instrukcji MOV, np.

```
MOV adres, #dana
```

gdzie adres jest z zakresu <0...31>.

Korzystanie z systemu banków rejestrów roboczych powoduje jednak, że listing programu, jest bardziej czytelny, a jego późniejsza analiza przez programistę szybsza.

## Układy czasowo-licznikowe

W procesorze 8051/C51 mamy do dyspozycji 2 takie układy (T0 i T1), a w kości 8052/C52 dodatkowo trzeci (T2). To, czym dokładnie są układy czasowo-licznikowe, dowiedzieliście się drodzy Czytelnicy z jednego z wcześniejszych odcinków klasy mikroprocesorowej. Zamieszczone tam informacje były jednak (przy braku znajomości języka asemblera 8051) dość teoretyczne. Warto jest jednak je sobie odświeżyć przed lekturą niniejszego paragrafu.

Teraz kiedy opanowaliśmy (choć może nie wszyscy w takim samym stopniu) sztukę bodaj prostego programowania procesora, będzie mi łatwiej zilustrować podane wcześniej wiadomości i sprowadzić je do czystej praktyki, wspartej jednak krótkimi, lecz niezbędnymi informacjami na temat układów czasowo-licznikowych.

Z układami czasowo-licznikovymi procesora 8051 związane są nierozłącznie dwa rejestry specjalne: **TCON** i **TMOD**.

Rejestr **TMOD** określa tryby pracy układu czasowo-licznikowego – zarówno dla T0 jak i T1. Rejestr ten nie jest adresowany bitowo. Wszystkie bity rejestru **TMOD** mogą być zmieniane wyłącznie programowo czyli przez użytkownika.

|       |                |                |      |
|-------|----------------|----------------|------|
| bity: | licznik T1     | licznik T0     |      |
| 89h   | GATE C/T M1 M0 | GATE C/T M1 M0 | TMOD |
|       | 7 6 5 4        | 3 2 1 0        |      |

Półowa rejestru (młodsze 4 bity) określa parametry układu licznika T0, natomiast 4 starsze bity określają to samo lecz dla układu licznikowego T1. Z tego względu przy opisie będę kierował się tylko jednym z liczników.

**GATE** – bit uaktywnienia zewnętrznego bramkowania licznika Tx (x=0,1). Kiedy GATE=0 to licznik pracuje wtedy kiedy bit TRx w słowie **TCON** jest ustawiony. Kiedy GATE=1 to licznik pracuje gdy TRx = 1 oraz wejście INTx (x:1 to INT1, x:0, to INT0) jest w stanie wysokim (INTx=1).

**C/T** – bit określający funkcję jaką pełni podczas pracy dany układ licznikowy, i tak gdy bit =0 to układ pracuje jako czasomierz taktowany wewnętrznym sygnałem zegarowym o częstotliwości Fxtal / 12. Gdy zaś bit = 1, to układ pracuje jako licznika impulsów zewnętrznych z wejścia Tx (T1 lub T0). Temat maksymalnej częstotliwości zliczanych impulsów zewnętrznych poruszany był w części 5 szkoły mikroprocesorowej.

**M1, M0** – bity określające wybór trybu pracy układu czasowo-licznikowego, i tak:

| M1 | M0 | Tryb | Opis                                                                                                                                                                                                           |
|----|----|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | 0  | 0    | 8-bitowy licznik THx taktowany za pośrednictwem 5-bitowego licznika TLx (x:1 to T1, x:0 to T0)                                                                                                                 |
| 0  | 1  | 1    | 16-bitowy licznik złożony z rejestrów THx, TLx                                                                                                                                                                 |
| 1  | 0  | 2    | 8-bitowy licznik TLx z automatycznym wpisywaniem wartości początkowej po przepełnieniu z THx                                                                                                                   |
| 1  | 1  | 3    | licznik T0 pracuje jako 2 niezależne 8-bitowe liczniki: TL0 jest sterowany za pomocą bitów sterujących licznika T0; TH0 jest sterowany za pomocą bitów sterujących licznika T1 licznik T1 nie pracuje w ogóle. |

Zajmiemy się teraz rejestrem **TCON**, w którym 4 najstarsze bity są bezpośrednio powiązane z układami czasowo-licznikovymi procesora. Oto on.

|       |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|
| bity: | 8Fh | 8Eh | 8Dh | 8Ch | 8Bh | 8Ah | 89h | 88h |
| 88h   | TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |
|       | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**TF1 (bit TCON.7, adres: 8Fh)** – znacznik przepełnienia licznika T1, jest sygnałem zgłoszenia przerwań. Ustawiany jest automatycznie – sprzętowo, zerowany także automatycznie przy przyjęciu przerwań.

## Też to potrafisz

**TR1 (bit TCON.6, adres: 8Eh)** — bit sterujący zliczaniem licznika T1. Gdy wyzerujemy go (TR1=0) to licznik się zatrzyma. Ustawienie (TR1=1) uruchamia licznik.

**TFO (bit TCON.5, adres: 8Dh)** — znacznik przepełnienia licznika T0, jest sygnałem zgłoszenia przerwania. Ustawiany jest automatycznie — sprzętowo, zerowany także automatycznie przy przyjęciu przerwania.

**TR0 (bit TCON.4, adres: 8Ch)** — bit sterujący zliczaniem licznika T0. Gdy wyzerujemy go (TR0=0) to licznik się zatrzyma. Ustawienie (TR0=1) uruchamia licznik.

Tak oto za pomocą dwóch rejestrów T0MD i TCON można sterować trybami i zachowaniem się liczników T0 jak i T1.

Jak w praktyce wykorzystuje się liczniki? Otóż podam kilka wskazówek i odpowiedzi które wynikają z codziennego „użytkowania” procesorów 8051 oraz różnorodności ich zastosowań. Dla ułatwienia będę operował symboliką związaną z licznikiem T0, pamiętając o tym że licznik T1 można traktować tak samo.

Na wstępie jedna ważna uwaga: **oba liczniki T0 i T1 zliczają tylko w górę!**

Tryb 16-bitowy licznika (tryb nr 1) wykorzystuje się często np. przy generowaniu przerwań związanych z odcinaniem czasu.

Jak wykorzystać licznik do generowania opóźnień w systemie lub do zmusić go do odcinania stałych dłuższych odcinków czasu? Ano należy wraz z licznikiem, wykorzystać związane z nim przerwanie — pojawiające się w momencie przepełnienia licznika.

W jaki sposób i dlaczego?

Popatrzmy. Skoro licznik ustawiony do pracy np. w trybie 16-bitowym (zliczanie impulsów wewnętrznych Fxtal / 12) to od jednego przepełnienia licznika do drugiego przepełnienia licznika minie czas określony zależnością:

$T = (10000h - \text{wartość początkowa TH0.TL0}) \times 12 / \text{Fxtal}$

gdzie Fxtal to częstotliwość rezonatora kwarcowego procesora.

Toteż przed uruchomieniem do rejestrów licznika T0 wpisujemy np. za pomocą instrukcji

```
MOV TH0, #....
```

```
MOV TH1, #....
```

wartość początkową.

Dzięki temu np. przy kwarcu 12MHz wartość 12 / Fxtal będzie równa 1 us (mikrosekundzie), co przy 16-bitowym liczniku da możliwość generowania opóźnień czasowych z przedziału od kilku mikrosekund do 65535 us, czyli prawie 65,5 milisekundy, z rastrem 1 us.

A co w przypadku chęci generowania większych niż 65 ms odstępów czasowych? Można oczywiście zmniejszyć częstotliwość procesora, ale w praktyce robi się to zupełnie inaczej. Otóż w procedurze obsługi przerwania z danego licznika, który przecież co pewien, ustalony przez programistę czas, przepełnia się, należy umieścić instrukcje rozszerzające zakres danego licznika o np. dodatkowe 8 bitów (jeden bajt). Wtedy uzyskamy licznik 24-bitowy, a to już daje spore możliwości. Jak to się robi praktycznie, przeczytacie w lekcji nr 8 w tym numerze.

Do generowania nadaje się także doskonale tryb z 8-bitowym licznikiem taktowanym za pośrednictwem 5-bitowego preskalera — tryb nr 0.

W trybach 0, 1 i 3 liczniki zarówno T0 jak i T1 po przepełnieniu należy ponownie załadować wartością początkową, w przeciwnym przypadku będą zliczały o zera.

Wyjątkiem jest tryb nr 2 z automatycznym ładowaniem wartości początkowej z rejestru TH0 do 8-bitowego licznika TL0 (to samo dotyczy licznika T1).

Tryb ten oprócz odcinania bardzo krótkich odcinków czasu (w trybie pracy licznika — jako czasomierza) ma dodatkowe zastosowanie do taktowania portu szeregowego w trybach: 1 i 3 (-> patrz artykuł w poprzednim numerze EdW).

Oto przykłady programowania wstępnego rejestrów układów czasowo-licznikowych dla różnych trybów pracy liczników. Wszystkie oznaczenia odnoszą się do licznika T0 ale można je także stosować dla licznika T1 w ten sam sposób.

a) licznik T0 16-bitowy pracujący jako czasomierz (T1, nie używany)

```
MOV TMOD, #0001b ;ustawienie trybu licznika
```

```
MOV TH0, #wartośćH ;wpisanie wartości
```

```
MOV TL0, #wartośćL ;początkowych
```

```
SETB TR0 ;i start licznika
```

Jak obliczyć wartości początkowe? Posłużę się przykładem. Otóż założmy że pracujemy z kwarcem 12MHz, czyli licznik zwiększa swoją wartość co 1 us.

Jeżeli zatem chcemy odcinąć np. 1 ms odstępów czasu (od jednego przepełnienia do drugiego przepełnienia ma upłynąć dokładnie 1 ms), to obliczamy wartość początkową licznika w sposób:

wartość początkowa = 65536 - (1 ms / 1 us) = 65536 - 1000 = 64536 = FC18h (szesnastkowo)

Zatem do rejestrów: TH0.TL0 należy wpisać wartość FC18h, np. za pomocą instrukcji:

```
MOV TH0, #FCh
```

```
MOV TL0, #18h
```

w przykładzie a) były to słowa wartości H i wartości L.

Sprawdźmy w teorii jak zadziałają instrukcje i kiedy licznik się przepełni, zapiszmy zatem:

```
MOV TMOD, #0001b ;ustawienie trybu
```

```
MOV TH0, #FCh ;wpisanie wartości
```

```
MOV TL0, #18h ;początkowej
```

```
SETB TR0 ;i start licznika
```

Kiedy licznik startuje jego wartość wynosi FC18h, czyli dziesiętnie 64536. Teraz licznik z każdą mikrosekundą będzie zwiększał zawartość aż do przepełnienia, czyli przekroczenia FFFFh. Stanie się to po dokładnie 3E8h (1000 dziesiętnie) cyklach zliczania czyli mikrosekundach. A przecież 1000 mikrosekund to 1 milisekunda, czyli wszystko jest w porządku. Nie należy tylko w przypadku chęci cyklicznego powtarzania odcinania takich samych odstępów czasu, zapomnieć o konieczności ponownego wpisywania wartości początkowej do liczników TH0 i TL0, zaraz po wejściu do procedury obsługi przerwania. Licznik bowiem po przepełnieniu nadal pracuje (zależy to tylko od stanu bitu TR0 w rejestrze TCON).

Wspomniana procedura może wyglądać następująco:

intT0:

```
MOV TH0, #FCh ;wpisanie wartości
```

```
MOV TL0, #18h ;początkowej
```

```
PUSH ;dalsze instrukcje
```

```
..... ;procedury obsługi przerwania
```

```
.....
```

```
.....
```

```
POP
```

```
reti
```

I tu powstaje pewna nieścisłość, otóż zauważyliśmy, że od czasu przepełnienia licznika do rozpoczęcia procedury obsługi przerwania oraz przeładowania zawartości TH0 i TL0 licznika, upływa zawsze trochę czasu procesora, a licznik bezustannie zlicza, tym razem od zera. Zatem aby uzyskane interwały czasowe pokrywały się z naszymi założeniami, należy uwzględnić zliczone do czasu przeładowania rejestrów TH0 i TL0, impulsy, korzystając z instrukcji ORL i przeładowywać rejestry licznika w sposób następujący:

intT0:

```
ORL TL0, #18h ;dodanie logiczne kilku
```

```
..... ;impulsów początkowych
```

```
MOV TH0, #FCh ;wpisanie wartości starszego
```

```
..... ;bajtu licznika
```

```
PUSH ;dalsze instrukcje
```

```
..... ;procedury obsługi przerwania
```

```
.....
```

```
POP
```

```
reti
```

Jednak i w tym przypadku błąd będzie wyeliminowany w przypadku, jeżeli od czasu przepełnienia do zgłoszenia przerwania nie upłynie więcej niż 7 cykli maszynowych procesora. Zauważmy, bowiem że instrukcja ORL (dodawania logicznego) sumuje poszczególne bity, a nie dodaje arytmetycznie dwie liczby. Zatem w przypadku liczby 18h, która może być zapisana binarnie jako:

00011000b

trzy najmłodsze bity są różne zero i dlatego zsumowanie będzie poprawne tylko z liczbą z zakresu 1...7. Dlatego przy programowaniu należy o tym pamiętać i ewentualnie tak definiować wartość początkową, aby wyeliminować błąd tzw. „pierwszych impulsów”. Ktoś powie, no dobrze, ale przecież można by dodać te liczby arytmetycznie używając instrukcji dodawania np. ADD, no tak ale po pierwsze angażujemy w to akumulator Acc, po drugie operacja dodawania ADD też trwa przed określoną liczbę cykli zegarowych, toteż niewiele to zmienia.

W każdym razie dłuższa praktyka i testowania generowanych przez was procedur pozwoli na dokładne zgłębienie problemu i znalezienie na niego niejednego rozwiązania. Przykłady takich rozwiązań podam przy najbliższej okazji w jednym z kolejnych odcinków szkoły mikroprocesorowej.

b) licznik zlicza impulsy określoną przez nas liczbę impulsów zewnętrznych z zakresu 1...255 a następnie sygnalizuje koniec zliczania wygenerowaniem przerwania. Wykorzystamy tryb 2 licznika gdzie TL0 pra-

nerowaniem przerwania. Wykorzystamy tryb 2 licznika gdzie TLO pracuje z automatycznym wpisywaniem wartości początkowej z TH0.

```
MOV TMOD, #0110b ;ustawienie trybu licznika
MOV TH0, #liczba ;wpisanie wart. początkowej
MOV TLO, TH0 ;konieczne !
SETB TR0 ;rozpoczęcie zliczania
```

przy czym wartość liczba jest równa:

liczba = 256 — liczba impulsów zewnętrznych do zliczenia

c) Tak na marginesie w podobny sposób można przystosować licznik T1 do taktowania portu szeregowego, oto sposób

```
MOV TMOD, #00100000b ;ustawienie trybu licznika T1
 ;(T0 stoi)
MOV TH1, #baud ;wpisanie prędkości transmisji
MOV TL1, TH1 ;tu nie jest konieczne
SETB TR1 ;i start taktowania
```

W miejsce „baud” należy wpisać liczbę określającą szybkość transmisji portu szeregowego. Wartości przykładowe oraz sposób obliczania znajduje się w poprzednim odcinku klasy mikroprocesorowej.

W każdym z przypadków zapisywaliśmy bezpośrednio rejestr TMOD określając sposób pracy układu czasowo-licznikowego T0 i T1. Jednak w pewnych przypadkach, w programie może zająć potrzeba modyfikacji trybu jednego licznika przy niezmiennym i niezakłóconej pracy drugiego licznika. Wtedy użycie komendy

```
MOV TMOD, #wartość
```

może okazać się dość ryzykowne.

Prostym rozwiązaniem jest maskowanie tej części bajtu TMOD która odpowiada za danych licznika, a którego nie chcemy zmieniać.

I tak elegancki zapis modyfikacji np. z przykładu c) będzie wyglądał następująco:

```
ANL TMOD, #0Fh ;licznika T0 nie ruszamy
ORL TMOD, #00100000b ;ustawienie trybu licznika T1,
 ;T0 bez zmian
MOV TH1, #baud ;wpisanie prędkości transmisji
MOV TL1, TH1 ;tu nie jest konieczne
SETB TR1 ;i start taktowania
```

W pierwszej komendzie listingu po prostu wyczyściliśmy najpierw 4 bardziej znaczące bajty rejestru TMOD (pamiętajmy że nie można adresować go bitowo), a następnie zapisaliśmy do nich odpowiedni tryb pracy licznika T1.

W taki sam sposób należy postępować z licznikiem T0, maskując wtedy bity licznika T1.

Pozostała nam jeszcze układ czasowo-licznikowy T2, który ze względu na wiele ciekawych funkcji dodatkowych (w porównaniu z T0 lub T12) omówię w kolejnym odcinku szkoły mikroprocesorowej.

Na razie zapraszam do lektury kolejnej lekcji nr 8.

Dzisiaj połączymy wiedzę z zakresu przerwań sprzętowych i liczników i przeanalizujemy procedurę realizującą funkcję zegara — czyli odmierzenia czasu rzeczywistego: sekundy, minuty, godziny, dni, miesiące a nawet lata.

**Sławomir Surowiński**

**Od redakcji.** Ze względu na ograniczoną objętość rubryki, a jednocześnie chęć zamieszczenia całego artykułu serii „Mikrokontrolery, to takie proste...”, kącik pocztowy w podwójnej objętości znajdzie się w następnym numerze EdW.

# Lekcja 8

W dzisiejszej lekcji zbierzemy nasze wiadomości dotyczące układu przerwań mikroprocesora oraz informacje przedstawione w dzisiejszym numerze EdW o układach czasowo-licznikowych.

Tematem lekcji będzie napisanie i wspólne przeanalizowanie procedury zliczania czasu rzeczywistego (sekund, minut i godzin) wykorzystującą przerwanie pochodzące od jednego z dwóch układów czasowo-licznikowych procesora 8051.

Połączenie właściwości zliczania wewnętrznych impulsów zegarowych przez układ licznikowy wraz z odpowiednim generowaniem przepełnienia tego licznika — czyli generowania przerwania pozwoli na dokładne odmierzenie sekund, a co za tym idzie minut oraz godzin.

Oprócz wspomnianej procedury, pracującej „w przerwaniu” (czyli okresowo) utworzymy także prościutki fragment programu pozwalający na wprowadzenie przez użytkownika czasu : godzin, minut i sekund, po czym po wciśnięciu dowolnego klawisza, uruchomienie zegara i rozpoczęcie zliczania czasu wraz z jego wyświetlaniem.

Tak powstały program można będzie załadować do komputerka i uruchomić. Ponieważ problem odmierzenia czasu spotykany jest bardzo często przy okazji układów mikroprocesorowych, niniejsza lekcja należy przestudiować bardzo uważnie, analizując wszystkie zawarte w niej komentarze oraz zamieszczony listing programu zegara.

Zrozumienie problemu implementacji zegara czasu rzeczywistego oraz właściwego generowania przerwań systemowych jest bowiem podstawą do dalszych, często przeprowadzanych samodzielnie eksperymentów.

## A oto założenia do programu:

1. W programie rezerwujemy 3 komórki w wew. pamięci RAM procesora, jedna będzie zliczać sekundy, druga minuty, trzecia godziny.
2. Zliczanie będzie odbywać się w kodzie BCD, czyli każdej pozycji liczby np. sekund będą odpowiadać 4 bity danej komórki pamięci, oto wyjaśnienie:  
— niech bajt zliczający sekundy nazywa się SEK, zdefiniujemy go jako np.  
`SEK equ 62h`

czyli w komórce wew. RAM procesora o adresie 62h będą zliczane sekundy czasu rzeczywistego z wykorzystaniem instrukcji korekty dziesiętnej akumulatora:

```
DA A
 ;czyli np. jeżeli licznik sekund będzie zawierał np.
09 (heksadecymalnie)
 to po inkrementacji — zwiększeniu o jeden
 powinien wskazywać (zgodnie z zapisem BCD)
10 (heksadecymalnie)
```

Wtedy przy użyciu procedury Bios a komputerka A2HEX (patrz opis z poprzednich lekcji klasy mikroprocesorowej) będzie można łatwo wyświetlić w czytelnej postaci aktualną wartość sekund. Podobnie postąpimy z minutami i godzinami. Wystarczy bowiem wydać komendy, np.:

```
mov A, SEK
mov B, #7
lcall A2HEX
```

aby na DL7 i DL8 pojawiła się aktualna wartość sekund — aktualna wartość licznika sekund SEK.

A co (lub kto) zajmie się inkrementacją sekund, minut i godzin? Właśnie procedura obsługi przerwania od jednego układu czasowo-licznikowego. W pętli głównej programu my będziemy troszczyć się jedynie o wyświetlanie na displayu komputerka wartości godzin, minut i sekund. W prosty sposób także wyświetlimy tzw. „migający dwukropek” w postaci kresków (myślników) pomiędzy pozycjami godzin i minut oraz minut i sekund w postaci:

DL 1 2 3 4 5 6 7 8

G G — M M — S S

gdzie: GG — pozycje godzin  
MM — pozycje minut  
SS — pozycje sekund

Np. godzina 12:34 i 57 sekund będzie wyświetlana jako:  
**12-34-57**



## Też to potrafisz

z migającymi znakami „-” (myślnika). Zauważcie że 8 pozycji wyświetlacza komputerka akurat wystarcza na wyświetlenie czasu w takiej właśnie formie (którą oczywiście należy traktować jako przykładową).

Korekta dziesiątka akumulatora po inkrementacji danej jednostki czasu (sekund, minut lub godzin) jest niezbędna, bowiem w przeciwnym przypadku po sekundach równych „09” nastąpiło by wyświetlenie wartości „0A”, a tego byśmy nie chcieli.

3. Do wygenerowania okresowo powtarzającej się procedury obsługi przerwania, w której będą odpowiednio inkrementowane komórki sekund, minut i godzin wykorzystamy układ czasowo – licznikowy T1 komputerka. Użycie licznika T0 jest niemożliwe, a przynajmniej nie na tym etapie nauki, ze względu na to że jest on już zajęty multiplexowym wyświetlaniem informacji na wyświetlaczu DL1...8 komputerka, dajmy więc mu spokój.
4. Dodatkowo w pętli głównej programu przed uruchomieniem zegara, dodamy kilka instrukcji dzięki którym będzie można wpisać aktualny czas – czyli po prostu „nastawić nasz zegarek”, a następnie go uruchomić.

### Wstępne obliczenia – wariant 1

W komputerku AVT-2250 procesor 8051 pracuje z częstotliwością rezonatora kwarcowego o wartości 11,0592 MHz, czyli

$$F_{xtal} = 11059200 \text{ Hz}$$

Zatem 1 cykl maszynowy procesora będzie trwał dokładnie:

$$T_m = 12 / F_{xtal} = 12 / 11059200 = 1,085069444 \mu s \text{ (mikrosekundy)}$$

Jak już wiesz, każdy z liczników procesora (T0, lub T1) pracując w trybie czasomierza zlicza wewnętrzne impulsy zegarowe w częstotliwością

$$F_{xtal} / 12 = 11059200 / 12 = 921600 \text{ Hz}$$

co jest dokładnie odwrotnością obliczonego wcześniej okresu cyklu maszynowego procesora  $T_m$ .

Zatem można powiedzieć żeby np. przepelnić licznik T1 co 1 sekundę i generować przez to przerwanie, trzeba by licznik ten zliczył:

$$n = 1 / T_m = F_{xtal} / 12 = 921600 \text{ impulsów}$$

Niestety, nawet w trybie 1, kiedy licznik pracuje jako 16-bitowy (tryb 1), jest w stanie zliczyć jedynie  $2^{16}-1$  impulsów, czyli 65535. Można zatem powiedzieć że licznik może się przepelnić najradziej co:

$$t = 65536 \times T_m = 71,111(1) \text{ ms (milisekund)}$$

a to stanowczo za mało. Cóż więc w tej sytuacji należy zrobić?

Odpowiedź na to pytanie jest prosta. Należy przepelnić licznik częściej niż co sekundę – np. **256 razy na sekundę** (z częstotl. 256 Hz), a w procedurze obsługi przerwania licznika wprowadzić dodatkową zmienną – licznik (bajt w wew. RAM procesora), który będzie inkrementowany (już bez korekty dziesiątnej) za każdym razem kiedy, nastąpi przepelnienie licznika. W ten sposób, w przypadku kiedy licznik ten będzie osiągał np. wartość maksymalną – 255 będzie to sygnałem że minęło właśnie **256 okresów po 1/256 sekundy**, co w efekcie oznacza że **minęła dokładnie 1 sekunda** i czas wobec tego zwiększyć licznik sekund (a co za tym idzie w razie potrzeby licznik minut i godzin). Prawda że logiczne, i tak też zrobimy!

Dlaczego wybrałem wartość 256 Hz do zliczania nazwijmy to „podsekund”, a nie np. 100 (to by było super zliczać także setne sekundy!). Tak to logiczne pytanie, tylko że w przypadku wartości rezonatora kwarcowego 11059200 Hz zliczanie 1/100 sekundy było by dość kłopotliwe, ze względu że ta wartość  $F_{xtal}$  nie dzieli się przez 12 i przez liczbę całkowitą aby dać właśnie 100.

Za to dzieli się przez 12 i przez 256 co w efekcie daje wartość:  $T_{1imp} = 3600$  (dziesiętnie) co w efekcie wyznaczy nam z podanej niżej zależności wartość początkowa licznika T1, która spowoduje że przepelnienie licznika nastąpi dokładnie po 1/256 sekundy.

$$TH1.TL1 = T_{1max} - T_{1imp} + 1 = 65535 - 3600 + 1 = 61936 = F1F0h \text{ (hexadec.)}$$

Zatem wartością początkową licznika przy rezonatorze 11059200 Hz i przy założonym okresie przepelniania równym 1/256 sek. jest liczba F1F0h, którą można zapisać do rejestrów SFR licznika za pomocą instrukcji np.

```
mov TH1, #0F1h
mov TL1, #0F0h
```

Wszystko było by dobrze, ale nie możemy zapomnieć o drobnym, aczkolwiek istotnym fakcie, a mianowicie, że od przyjęcia przerwania do każdorazowego przeładowania licznika w procedurze przyjęcia przerwania mija bliżej nieokreślona liczba cykli maszynowych, w których licznik ciągle zlicza impulsy po przepelnieniu – czyli od wartości 0000h. Wprawdzie można policzyć ile cykli maszynowych przez ten czas, ale trzeba znać wszystkie rozkazy które znajdują się „po drodze”, czyli:

- a) cykle od przepelnienia licznika do przyjęcia przerwania – w praktyce jest ich 2 (w przypadku kiedy przerwanie ma najwyższy priorytet, lub nie trwa obsługa przerwania o wyższym priorytecie);
- b) cykle potrzebne na skok do tablicy wektorów przerw – w przypadku licznika T1 procesor automatycznie wykona skok pod adres podany w artykule:

001Bh

Pod tym adresem powinien znajdować się skok do właściwej procedury obsługi przerwania w postaci instrukcji np. :

```
ljmp intT1
```

no tak ale gdzie fizycznie jest ta etykieta – ten adres?

Przecież nie można znaleźć się w obszarze zewnętrznej pamięci RAM procesora, w którym znajduje się zawarty w EPROM-ie monitor komputerka AVT-2250. Nie można bowiem „w miejsce” w którym znajduje się Bios zawarty w EPROM-ie wpisać instrukcji naszego programu obsługi zegara. Co zatem zrobić, czyżby nie dało się jakoś ujarzmić przerwanie i przekazać jego wektora w obszar zewnętrznej pamięci operacyjnej komputerka – czyli w miejsce gdzie ładowany jest kod programu użytkownika – pamięć SRAM? Można.

Konstruktor komputerka, czyli Ja przewidziałem taką możliwość i postanowiłem w prosty sposób „wyprowadzić” wszystkie wektory przerwania z pamięci Bios’a komputerka w obszar pamięci SRAM, w miejsce ustalone dodatkowo przez użytkownika, a to ci dopiero gratka!

Aby wyjaśnić to przypomnę że tabela wektorów przerw dla 8051 przedstawia się następująco:

| Adres | Opis                           |
|-------|--------------------------------|
| 0003h | przerwanie z wejścia /INT0     |
| 000Bh | przerwanie z licznika T0       |
| 0013h | przerwanie z wejścia /INT1     |
| 001Bh | przerwanie z licznika T1       |
| 0023h | przerwanie z portu szeregowego |

W programie Bios a komputerka w miejscu każdego wektora znajduje się skok typu LJMP do tzw. „procedury inicjującej przerwanie” z której to dopiero następuje skok do właściwego miejsca w zewnętrznej pamięci SRAM – operacyjnej.

Wspomniana procedura inicjująca (nie dotyczy to licznika T0) jest tak zbudowana, że powoduje ona skok pod adres w zewn. SRAM komputerka pod adres, którego:

- młodszy bajt (pogrubione w tabeli) nie zmienia się
- starszy bajt jest „brany” z komórki w wew. RAM procesora o adresie 72h – (patrz opis Bios’a komputerka)

Komputerowcy mogą w tym miejscu zajrzeć do zbioru „CONST.INC” na dyskiecie AVT-2250/D i sprawdzić deklarację

```
intvec equ 72h
```

która potwierdza te założenie.

Zatem reasumując jeżeli na początku naszego przykładowego programu przed uruchomieniem układu przerwania od licznika T1 wpisemy do tej komórki (wew. RAM!) liczbę np. 80h, to tabel wektorów przerw procesora 8051 zostanie niejako „wyprowadzona” do obszaru o adresach jak poniżej:

| Adres | Opis                           |
|-------|--------------------------------|
| 8003h | przerwanie z wejścia /INT0     |
| 800Bh | przerwanie z licznika T0       |
| 8013h | przerwanie z wejścia /INT1     |
| 801Bh | przerwanie z licznika T1       |
| 8023h | przerwanie z portu szeregowego |

Bardziej zaawansowani i wnikliwi czytelnicy zauważą w tym miejscu ciekawy fakt, mianowicie, że takie postępowanie Bios a komputerka umożliwia kontrolowanie wszystkich źródeł przerwania a nawet zablokowanie wyświetlacza (który pracuje na przerwaniu od licznika T0) i wykozystanie go do własnych celów, czego na razie stanowczo odradzam.

Podam liczbę **80h** nie przypadkowo, bowiem od tego adresu – 8000h na płycie komputerka (przy założeniu że zworka JP3 jest w pozycji 8000h) zaczyna się pamięć operacyjna gdzie ładowany będzie program.

Podsumowując prześledźmy co fizycznie się stanie w przypadku przepelnienia licznika T1:

- zgłoszone zostaje przerwanie
- procesor skacze do „pierwotnej” tablicy wektorów przerw, czyli pod adres 001Bh; jest to jednakże obszar pamięci EPROM – Bios-u, gdzie zawarta jest instrukcja:

```
LJMP intT1
```

czyli skoku do etykiety intT1, która to znowu etykieta znajduje się także w obszarze Bios a komputerka a za nią znajdują się instrukcje

```
servT1:
```

```
push Acc
push DPH
push DPL
clr A
mov DPH,intvec ;pobranie zewn. wektora T1
mov DPL,#1Bh
jmp @A+DPTR
```



Zadaniem tych instrukcji jest zapamiętanie modyfikowanych w procedurze przerwania rejestrów – są to Akumulator i rejestr DPTR (DPH i DPL), a następnie wykonanie skoku bezwarunkowego (ostatnia instrukcja) pod adres będący sumą zawartości akumulatora (równy 0) oraz wskaźnika DPTR. Zanim to jednak następuje, wskaźnika DPTR jest ładowany wspomnianym wcześniej adresem będącym „złożeniem” starszego bajtu (DPH) równego zmiennej „intvec” (adres 72h), którą modyfikuje użytkownik – w naszym przypadku będzie to 80h, oraz młodszego bajtu będącego odpowiednikiem pierwotnej tabeli wektorów przerwań, czyli 1Bh. W sumie procesor wykona skok pod adres: 801Bh, gdzie powinna znajdować się napisana przez nas procedura obsługi przerwania od licznika T1, a obsługująca zliczanie czasu rzeczywistego.

Uff, trochę to skomplikowane, ale niestety niezbędne, bowiem w przypadku korzystania z zestawów edukacyjnych często z zawartym w nich mniej lub bardziej skomplikowanym Bios-em (a do takich należy AVT-2250) tak procedura jest konieczna. W przyszłości w autonomicznych układach opartych o 8051 i podobne, a nie wykorzystujących naszego Bios-a komputerka, przedstawione kroki są do pominięcia. Procesora po prostu skoczy do pierwotnej tabeli wektorów przerwań a następnie wykona skok do właściwego miejsca w Twoim programie, tam gdzie znajduje się procedura obsługi danego przerwania.

Wracając do tematu zauważmy jednak, że procesor na tych kilku etapach od przepełnienia licznika do skoku wreszcie do właściwej procedury obsługi przerwania potrzebować będzie prawdopodobnie **kilkunastu cykli procesora**, podczas (jeszcze raz powtarzam) **pracuje licznik T1!**

I to właśnie może stać się powodem błędu w dokładnym okresowym (co 1/256 sek) generowaniu przepełnienia licznika T1 – i co za tym idzie powstania przerwania.

Z grubsza można policzyć, (na podstawie tabeli instrukcji z wkładki EdWV) że zanim procesor przeładuje licznik w procedurze przerwania, to licznik zdąży już zliczyć od 0000h mniej więcej 17 impulsów – można to policzyć analizując instrukcje z ostatniego listingu od etykiety „**intT1**”.

Mogą nie pomóc instrukcje uwzględniający ten fakt, o których wspominałem w artykule przed niniejszą lekcją typu:

```
orlTL1, #...
```

```
orlTL1, #....
```

```
mov TH1, #....
```

pomóc jedynie może i to z doskonałym skutkiem inny tryb pracy licznika T1 a mianowicie **tryb 0**.

Jak pamiętasz w trybie tym licznik pracuje jako 8-bitowy (liczy TH1), a sygnał zegarowy (Fxtal / 12) jest dzielony dodatkowo przez 5-bitowy prescaler (czyli w praktyce przez 32) czyli rejestr TL1.

Dla nas i naszych kłopotów oznacza to tylko jedno – wybawienie, bowiem fakt, że do licznika TH1 „trafia” co trzydziesty drugi impuls (przez prescaler dzielnik TL1) pozwoli nam na uniknięcie wspomnianego błędu – kilkunastu cykli zegarowych od zgłoszenia przerwania do jego przyjęcia i przeładowania licznika T1.

Po prostu w czasie kiedy będą wykonywane te „wszystkie skoki” z jednej tablicy wektorów do drugiej a potem do właściwej procedury obsługi przerwania (o których mówiłem wcześniej) licznik nie zdąży zliczyć ani jednego impulsu – i oto chodzi.

I choć w teorii wydaje się to niepotrzebną komplikacją, w tym praktyce tryb 0 licznika jest najbardziej wygodnym i pewnym, jeżeli chodzi o generowanie opóźnień niezbędnych do odmierzenia czasu – szczególnie rzeczywistego. Niestety musimy w tym celu zmienić nieco nasze obliczenia.

### Wstępne obliczenia – wariant 2

Korzystamy z trybu 0 licznika T1. W tym trybie pracuje połowa licznika a mianowicie TH1, który może zliczyć maksymalnie 255 impulsów. Jednak częstotliwość tych impulsów będzie mniejsza niż w wariantie 1, bowiem przed licznikiem TH1 znajduje się 5-bitowy prescaler czyli nic innego jak dzielnik przez 32. Wobec tego częstotliwość impulsów zliczanych przez nasz licznik TH1 będzie wynosiła:

$$fz = Fxtal / 12 / 32 = 28000 \text{ Hz}$$

Ponieważ podobnie jak w wariantie 1, liczba ta przekracza aktualną pojemność licznika – tym razem 8-bitowego TH1, należy zastosować licznik pośredni na zasadach takich jak poprzednio. Załóżmy że stopień podziału będzie taki sam czyli 256, ale wtedy fz nie podzieli się przez 256, bowiem :

$$fz / 256 = 28800 / 256 = 112,5$$

czyli nie jest liczbą całkowitą, a to jest niedopuszczalne. Przyjmijmy zatem podział pośredni jako mniejszy o rząd (w kodzie dwójkowym o 2), będzie to zatem 128.

Wtedy :

$$fz / 128 = 28800 / 128 = 225 (= TH1imp)$$

Podsumowując, można powiedzieć, że jeżeli licznik TH1 zliczy za każdym razem 225 impulsów o częstotliwości fz (28800 Hz) to najmniej mu to dokładnie 1/128 sekundy. Jeżeli do tego dodatkowo – pośredni licznik zliczy te 128 „ułamków sekundy” to w sumie będziemy mieli odmierzoną pełną sekundę! I o to właśnie chodzi.

Pozostaje jeszcze jeden drobiazg, mianowicie obliczenie wartości początkowej licznika T1 na podstawie obliczonej liczby impulsów które powinienn zliczyć do przepełnienia, będzie to zatem:

$$TH1pocz = TH1max - TH1imp + 1 = 256 - 225 + 1 = 31$$

I taką właśnie wartość należy wpisywać do licznika TH1 za każdym razem po jego przepełnieniu. Wnikliwy czytelnik może szybko przeanalizować obliczenia z odwrotnej strony na podstawie formuły:

$$(256 - TH1pocz) \times 128 \times 32 \times (12 / Fxtal) = 1 \text{ sekunda}$$

nie mniej nie więcej! (pamiętaj: Fxtal = 11059200 Hz)

Na podstawie tych rozważań, można zabrać się do pisania programu. Listing poniżej przedstawia cały program wraz z procedurą obsługi przerwania. Każda linia, znana już zarówno komputerowcom jak i ręczniakom, jest poprzedzona numerem linii, dzięki czemu będzie mi łatwiej tłumaczyć po krótko każdą z nich. Uwaga, linie komentarzy będę pominął, a więc zaczynamy (ręczniacy mogą zacząć równocześnie wklepywać kod programu umieszczony w kolumnie trzeciej).

Linie 18...20 : na początku definiuję adresy komórek zliczających sekundy, minuty i godziny, ot tak sobie zająłem wolne komórki od adresu 60h do 62h.

Dodatkowo z linii 22 definiuję wspomniany pośredni licznik, którego zadaniem będzie zliczanie przepełnień licznika TH1.

W linii 26 definiuję wartość początkową licznika TH1 jako wyrażenie „Czest”.

Program zaczyna się w linii 30 deklaracją „ORG 8000h”, czyli że program jak zwykle umieszczamy od adresu podanego po tej deklaracji.

Aby ominąć występującą za „kilka adresów” zewnętrzną tabelę wektorów przerwań, w linii 31 umieszczam skok bezwzględny do etykiety START, gdzie rozpoczyna się właściwy program.

No i wreszcie dyrektywa „ORG 801Bh” w linii 35 definiuje mi adres od którego spokojnie będę pisał procedurę obsługi przerwania od licznika T1.

W linii 36 zaczyna się procedur obsługi przerwania – intT1. Pierwsze co należy koniecznie zrobić, to (w linii 37) przeładować zawartość licznika T1 – TH1, co czynię.

Dalej jak widać brak postulowanych instrukcji odkładania na stos modyfikowanych rejestrów, bowiem zostały one już zapamiętane na stosie podczas przyjęcia przerwania w procedurze pośredniej w obszarze Bios-a komputerka (listing wcześniej) > Dla przypomnienia powiem że odłożono w kolejności rejestry:

```
push Acc
```

```
push DPH
```

```
push DPL
```

W liniach 39...42 inkrementuję komórkę zliczającą ilość przepełnień licznika TH1, a następnie porównuję jej zawartość z zerem (w końcu obojętne czy jest to zero, czy 255, bo i tak każda z wartości pojawia się raz na 128). Jeżeli warunek jest spełniony, to znaczy że minęło 128 przepełnień licznika TH1, czyli w praktyce minęła 1 sekunda. Jeżeli tak się stanie to dzięki instrukcji w linii 42 program procesor skoczy do linii 44, w przeciwnym przypadku do na koniec procedury obsługi przerwania – linia 65, etykieta „koniecT1”.

Załóżmy więc że minęła sekunda, program kontynuowany jest od linii 44, gdzie do akumulatora ładowana jest zawartość licznika sekund.

Następnie w linii 45 jest ona inkrementowana, a w linii 456 zgodnie z naszym założeniem zliczania z kodzie BCD następuje korekta dziesiątnej akumulatora (uwaga, w tym miejscu – linia 45 – nie można zastosować instrukcji „INC” bowiem nie „współpracuje” ona z instrukcją korekty dziesiątnej „DA A”).

W linii 47 powiększona zawartość sekund zostaje przepisana z Acc do SEK, a następnie w linii 48 następuje porównanie, czy aby licznik sekund nie przekroczył wartości 59h (59 sekund?). Jeżeli tak nie jest procedura kończy się i następuje skok na jej koniec – do etykiety „koniecT1”.

W przeciwnym przypadku w linii 49 licznik sekund jest zerowany, a dalej w liniach 51...54 następuje korekta licznika minut w sposób identyczny jak w przypadku sekund.

W linii 55 licznik minut jest sprawdzany i w przypadku przekroczenia wartości 59 minut, następuje w linii 56 jego wyzerowanie i zostaje wykonana korekta licznika godzin – linie 58...61.

Podobnie dzieje się z licznikiem godzin, z tym że w linii 62 porównuje się jego zawartość z liczbą 24h. Jeżeli godzina 23-cia został przekroczona, to licznik godzin zeruje się w linii 63, i zaczyna się nowa doba.

W tym miejscu za linią 63 można by dopisać zliczanie dni tygodnia, dni, miesiący a nawet lat. To ciekawy temat na zadanie dla Was drodzy Czytelnicy. Przypominam tylko o fakcie istnienia nieregularnej ilości dni w kolejnych miesiącach roku, oraz istnieniu lat przestępnych.

Procedura obsługi przerwania ma się ku końcowi w linii 65, gdzie dalej w liniach 66...68 następuje odtworzenie zmodyfikowanych wcześniej rejestrów Acc i DPTR w odwrotnej kolejności niż przy odkładaniu na stos (zgodnie z zasadą przechowywania danych na stosie!).

Na koniec w linii 69 występuje instrukcja RETI, która jest konieczna do prawidłowego zakończenia przyjęcia przerwania.

# Też to potrafisz

Od linii 71 zaczyna się część główna programu.  
Analizę tej, dość prostej, części programu, pozostawiam Wam jako prace domową.

## Zadanie 1

Przeanalizować teoretycznej przebieg części głównej programu na podstawie linii

72...125, a następnie sprawdzić to w praktyce ładując program do komputerka i uruchamiając go.

## Zadanie 2

Zmodyfikować wyświetlanie czasu do postaci np. dla godziny 12:34 i 58 sekund

1 2 – 3 4.5 8

## Listing

```
1 CPU '8052.DEF'
2 ;*****
3 ;Klasa mikroprocesorowa – LEKCJA 8
4 ;Program obsługi zegara czasu rzeczywistego
5 ;na komputerkach edukacyjnych AVT-2250
6 ;*****
7 ;procedura korzysta z przerwania licznika T1 (tryb 0)
8 ;wykorzystywane są 3 komórki wewn.RAM procesora
9 ;zegar liczy: godziny, minuty i sekundy
10 ;w trybie 24-godzinny
11 ;*****
12 include 'const.inc'
13 include 'bios.inc'
14
15 ;Definicje komórek w wewn.RAM procesora
16 ;zajmowane przez dane zegara
17
18 0060 GODZ equ 60h ;licznik godzin
19 0061 MIN equ 61h ;licznik minut
20 0062 SEK equ 62h ;licznik sekund
21
22 0063 licz128 equ 63h ;licznik 1/128 sek
23
24 ;Definicje stałych wykorzystywanych w programie
25
26 001F Czyst equ 31 ;wartosc pocz.
 ;licznika TH1
27
28 ;*****
29 ;Początek kodu programu
30 8000 org 8000h
31 8000 028054 jmp START ;petla glowna od
 ;etyk. START
32
33 ;*****
34 ;Wektor przerwania od licznika T1
35 801B org 801Bh
36 801B ;początek proc.przer.T1
37 801B 758D1F mov TH1,#Czyst ;przeladowanie
 ;licznika T1
38 801E
39 801E 0563 inc licz128 ;zwiększenie
 ;licznika 1/128sek.
40 8020 E563 mov A,licz128
41 8022 C2E7 clr Acc.7
42 8024 7027 jnz koniecT1
43
44 8026 E562 mov A,SEK
45 8028 2401 add A,#1 ;zwiększenie
 ;licznika sekund
46 802A D4 da A ;z korekcja
 ;dziesiętna
47 802B F562 mov SEK,A
48 802D B4601D cjne A,#60h,koniecT1 ;czy SEK > 59?,
 ;nie to skocz
49 8030 756200 mov SEK,#0 ;tak to zeruj
 ;sekundy
 ;i koryguj minuty
50 8033
51 8033 E561 mov A,MIN
52 8035 2401 add A,#1 ;zwiększenie
 ;licznika minut
53 8037 D4 da A ;z korekcja
 ;dziesiętna
54 8038 F561 mov MIN,A
55 803A B46010 cjne A,#60h,koniecT1 ;czy MIN > 59?,
 ;nie to skocz
56 803D 756100 mov MIN,#0 ;tak to zeruj minuty
57 8040 ;i koryguj godziny
58 8040 E560 mov A,GODZ
59 8042 2401 add A,#1 ;zwiększenie
 ;licznika minut
60 8044 D4 da A ;z korekcja
 ;dziesiętna
61 8045 F560 mov GODZ,A
62 8047 B42403 cjne A,#24h,koniecT1 ;czy GODZ > 23 ?,
 ;nie to skocz
63 804A 756000 mov GODZ,#0 ;tak to zeruj
 ;godziny
64
65 804D koniecT1:
66 804D D082 pop DPL ;odtworzenie
 ;rejestrów
67 804F D083 pop DPH ;ze stosu
```

## Zadanie 3

Bardziej zaawansowanym i cierpliwym polecam uzupełnienie procedury obsługi przerwania (zegara) o obliczanie daty : dzień – miesiąc – rok (bez uwzględniania lat przestępnych) oraz umożliwienie wyświetlania tej daty na wyświetlaczu z możliwością przełączenia na czas i odwrotnie za pomocą klawisza.

Rozwiązania zadań 1 i 2 w kolejnej lekcji szkoły mikroprocesorowej. Najciekawsze a co najważniejsze poprawne propozycje rozwiązania zadania 3 przedstawię na łamach EdW w ramach „Skrzynki porad 8051”.

Sławomir Surowiński

```
68 8051 D0E0 pop Acc
69 8053 32 reti
70
71 8054 ;*****
72 8054 C28E START:
73 8056 53890F clr TR1 ;licznik T1 stop
 ;wyczyszczenie
 ;bitów T1
 ;T1 jako 16-bitowy
 ;(tryb 1)
74 8059 438900 orl TMOD,#00h ;załadowanie
 ;licznika
75 805C 758D1F mov TH1,#Czyst ;wyzierowanie
 ;licznika 1/256 sek.
76 805F 756300 mov licz128,#0 ;załadowanie MSB
 ;wektora przerwan
77 8062 757280 mov intvec,#80h ;odblokowanie
 ;przerwania od T1
78 8065 D2AB setb ET1 ;priorytet na to
 ;przerwanie
79 8067 D2BB setb PT1
80
81 8069 120274 lcall CLS ;wyczyszczenie
 ;displeja
82 806C 757840 mov DL1,#_minus
83 806F 757940 mov DL2,#_minus
84 8072 75F001 mov B,#1
85 8075 1203A7 lcall GETACC ;pobranie
 ;początkowej
 ;godziny
86 8078 F560 mov GODZ,A
87
88 807A 757B40 mov DL4,#_minus
89 807D 757C40 mov DL5,#_minus
90 8080 75F004 mov B,#4
91 8083 1203A7 lcall GETACC ;pobranie
 ;początkowej
 ;minuty
92 8086 F561 mov MIN,A
93
94 8088 757E40 mov DL7,#_minus
95 808B 757F40 mov DL8,#_minus
96 808E 75F007 mov B,#7
97 8091 1203A7 lcall GETACC ;pobranie
 ;początkowej
 ;sekundy
98 8094 F562 mov SEK,A
99 8096
100 8096 757A40 mov DL3,#_minus ;zapalenie kresk
 ;w postaci
101 8099 757D40 mov DL6,#_minus ;GG-MM-SS
 ;(godzina
 ;wprowadzona !)
102 809C 74FA mov A,#250
103 809E 120295 lcall DELAY ;odroczenie
 ;ok. 0,5 sekundy
104 80A1 1202C5 lcall CONIN ;czekanie na start
 ;zegara (klawisz)
105 80A4 D28E setb TR1 ;start licznika
 ;(zegara)
106
107 80A6 pokaz:
108 80A6 E563 mov A,licz128
109 80A8 30E608 jnb Acc.6,pelne ;co 1/2 sekundy
 ;pokazuj na zmianie
 ;puste DL3 i DL6
110 80AB 757A00 mov DL3,#0
111 80AE 757D00 mov DL6,#0
112 80B1 8006 sjmp czas
113 80B3 757A40 pelne: mov DL3,#_minus ;i kreski na DL3
 ;i DL6
114 80B6 757D40 czas: mov DL6,#_minus
115 80B9
116 80B9 E560 mov A,GODZ
117 80BB 75F001 mov B,#1 ;na DL1.DL2
 ;wypisz godziny
118 80BE 12024E lcall A2HEX
119 80C1 E561 mov A,MIN
120 80C3 75F004 lcall B,#4 ;na DL4.DL5
 ;wypisz minuty
121 80C6 12024E mov A,SEK
122 80C9 E562 mov B,#7 ;na DL6.DL7
 ;wypisz sekundy
123 80CB 75F007 lcall A2HEX
124 80CE 12024E sjmp pokaz ;i od początku
125 80D1 80D3
126
127 80D3 END
```



W kolejnym odcinku naszego cyklu zapoznamy się z pracą dodatkowego układu czasowo licznikowego, który występuje w mikrokontrolerach 8052/C. W drugiej części artykułu przedstawię specjalne tryby pracy procesorów, dzięki którym możliwe jest konstruowanie energooszczędnych autonomicznych układów bateryjnych. I choć na tym etapie nauki panowania nad mikrokontrolerem 8051 i jemu podobnymi, za wcześnie na podawanie przykładowych rozwiązań konstrukcyjnych, to tę część teorii warto poznać, zanim zabierzemy się do budowania „inteligentnych urządzeń elektro-nicznych”... a raczej mikroelektro-nicznych.



Poprzedni odcinek szkoły mikroprocesorowej poświęciłem omówieniu strony praktycznej układów czasowo-licznikowych T0 i T1 procesora 8051/52. Podałem też praktyczne wskazówki na powiązanie pracy liczników z układem przerwań procesora. Dzięki takiemu połączeniu udało nam się wspólnie stworzyć i przeanalizować przykładowy programik odmierzający czas. To podstawa do zrozumienia prawidłowego programowania mikrokontrolerów, bowiem prawie 100% programów pisanych na procesory wykorzystuje procedury (podprogramy) obsługi przerwań do kontroli pewnych funkcji samego procesora jak i układów peryferyjnych.

Umiejętne zaprogramowanie układu licznikowego oraz układu przerwań gwarantuje sukces w działaniu programu a dodanie funkcji zabezpieczających – tzw. „programowych haczyków” (których omówieniem zajmę się przy okazji następnego odcinka klasy mikroprocesorowej) gwarantuje poprawną pracę mikrokontrolera nawet w sytuacjach z góry nieprzewidzianych, jak np. przypadkowe przeładowanie liczników, czy automatyczne rozpoznawanie prędkości transmisji szeregowej z urządzenia zewnętrznego. Wróćmy jednak do tematu i zajmijmy się układem czasowo-licznikowym (potocznie nazywanym licznikiem) T2.

## TIMER T2 w 8052/C

Jak powiedziałem wcześniej, układ ten nie występuje w procesorach 8051/C51, jest natomiast integralną częścią struktury procesorów 8052/C. Z pewnych względów przy niektórych aplikacjach dwa układy czasowo-licznikowe (T0 i T1) to za mało, wtedy warto sięgnąć właśnie po kostkę '52.

Licznik T2 podobnie jak poprzednie T0 i T1 jest 16-bitowy, a zliczanie odbywa się przy pomocy dwóch 8-bitowych rejestrów z grupy SFR o adresach:

**0CDh:** TH2 – starsze 8 bitów licznika T2  
**0CCh:** TL2 – młodsze 8 bitów licznika T2

Rejestry operacyjne licznika T2:

|        |            |           |           |           |           |           |           |           |           |        |            |
|--------|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|--------|------------|
| nazwa: | <b>TH2</b> | <b>d7</b> | <b>d6</b> | <b>d5</b> | <b>d4</b> | <b>d3</b> | <b>d2</b> | <b>d1</b> | <b>d0</b> | adres: | <b>CDh</b> |
| nazwa: | <b>TL2</b> | <b>d7</b> | <b>d6</b> | <b>d5</b> | <b>d4</b> | <b>d3</b> | <b>d2</b> | <b>d1</b> | <b>d0</b> | adres: | <b>CCh</b> |

Tak jak poprzednio rejestry te można odczytywać jak i modyfikować, przeładowując (zmieniając) ich zawartość np. za pomocą instrukcji:  
mov TH2, #wartH  
mov TL2, #wartL

Z układem czasowo-licznikowym T2 związane są dwa dodatkowe 8-bitowe rejestry, (tworzące 16-bitowy rejestr **RLD**) a mianowicie:

**0CBh:** RLDH – starsze 8 bitów rejestru RLD  
**0CAh:** RL DL – młodsze 8 bitów rejestru RLD

Rejestry dodatkowe licznika T2:

|        |             |           |           |           |           |           |           |           |           |        |            |
|--------|-------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|--------|------------|
| nazwa: | <b>RLDH</b> | <b>d7</b> | <b>d6</b> | <b>d5</b> | <b>d4</b> | <b>d3</b> | <b>d2</b> | <b>d1</b> | <b>d0</b> | adres: | <b>CBh</b> |
| nazwa: | <b>RLDL</b> | <b>d7</b> | <b>d6</b> | <b>d5</b> | <b>d4</b> | <b>d3</b> | <b>d2</b> | <b>d1</b> | <b>d0</b> | adres: | <b>CAh</b> |

Rejestr RLD (RLDH.RL DL) pełni dwojaką rolę w zależności od trybu pracy licznika T2. Po pierwsze może być rejestrem wartości początkowej licznika T2 (TH2.TL2). Wtedy to w trybie pracy, który za chwilę omówię, po przepełnieniu licznika, wartość początkowa tego licznika zostaje automatycznie (bez programowego przeładowywania) przepisana z RLDH do TH2 oraz z RL DL do TL2.

W innym przypadku rejestr RLD może pełnić rolę rejestru zatraskowego, w którym zapamiętywana jest zawartość licznika T2 (TH2.TL2) w pewnych szczególnych momentach, o tym także za chwilę.

Podobnie jak omawiane na poprzednim odcinku układy T0 i T1, licznik T2 może pełnić dwie różne funkcje:

- może pracować jako **czasomierz**, czyli zliczać impulsy wewnętrzne o częstotliwości równej częstotliwości oscylatora Fxtal podzielonej przez 12 (zastosowanie takiego trybu pracy już poznaliśmy na przykładzie z lekcji nr 8)
- może także **zliczać impulsy zewnętrzne** z specjalnego wejścia T2, którym jest pin 1 procesora (alternatywna funkcja bitu 0 portu P1). Zwiększenie zawartości licznika T2 następuje w tym przypadku w momencie wykrycia zbocza opadającego na wejściu T2 (P1.0). Z licznikiem T2 związane jest także dodatkowe wejście T2EX – końcówka procesora P1.1 (pin 2). Końcówka ta może pełnić rolę sygnału strobującego zatrzaśnięcie zawartości licznika T2 (TH2.TL2) w rejestrach RLD (RLDH.RL DL), w innym przypadku umożliwia zdalne załadowanie wartości początkowej z rejestrów RLD do rejestrów licznika T2.

We wszystkich trybach pracy licznika T2 zasada jego pracy jest podobna jak dla układów T0 i T1. A więc obowiązują zasady dotyczące np. wykrywania stanów niskich i wysokich (na przemian) w przypadku zliczania impulsów zewnętrznych, ich częstotliwości maksymalnej, w zależności



od częstotliwości oscylatora procesora. Wszystkie omówiłem w poprzedniej części klasy mikroprocesorowej, warto więc je sobie przypomnieć.

Warto także wspomnieć o możliwości pracy licznika T2 w trybie taktowania portu szeregowego. Wtedy licznik T1 może być użyty do innych celów, natomiast T2 ze względu na swoją 16-bitową „naturę” pozwala na generowanie bardzo niskich częstotliwości taktowania portu, a dlaczego, o szczegółach powiem za chwilę.

Główny rejestrem sterującym trybami oraz pracą licznika T2 jest rejestr T2CON (SFR adres 0C8h). Z pewnością pamiętasz drogi Czytelniku, że w przypadku liczników T0 i T1 był to rejestr TMOD (oraz częściowo TCON), natomiast w przypadku układu czasowo-licznikowego T2 wszystkimi funkcjami licznika T2 steruje tylko jeden rejestr T2CON.

|              |            |             |             |             |              |            |             |               |            |
|--------------|------------|-------------|-------------|-------------|--------------|------------|-------------|---------------|------------|
| bity:        | CFh        | CEh         | CDh         | CCh         | CBh          | CAh        | C9h         | C8h           |            |
| <b>T2CON</b> | <b>TF2</b> | <b>EXF2</b> | <b>RCLK</b> | <b>TCLK</b> | <b>EXEN2</b> | <b>TR2</b> | <b>C/T2</b> | <b>CP/RL2</b> | <b>C8h</b> |
|              | 7          | 6           | 5           | 4           | 3            | 2          | 1           | 0             |            |

Rejestr jest adresowany bitowo (w odróżnieniu od TMOD w przypadku T0 i T1), co znacznie ułatwia operacje modyfikacji poszczególnych jego bitów. Oto ich znaczenie:

**TF2 (bit T2CON.7, adres: CFh)** – bit ustawiany w momencie przepełnienia licznika T2, jest znacznikiem zgłoszenia przerwania przez T2

**EXF2 (bit T2CON.6, adres CEh)** – bit ustawiany w momencie wykrycia opadającego zbocza na wejściu T2EX (pin 2 procesora), aktywny gdy bit EXEN2 = 1, jest sygnałem zgłoszenia przerwania

**RCLK (bit T2CON.5, adres CDh)** – ustawienie tego bitu powoduje przyłączenie licznika T2 jako taktującego (przepełnieniem) odbiornika portu szeregowego procesora

**TCLK (bit T2CON.4, adres CCh)** – ustawienie tego bitu powoduje przyłączenie licznika T2 jako taktującego (przepełnieniem) nadajnika portu szeregowego procesora

**EXEN2 (bit T2CON.3, adres CBh)** – bit uaktywniający wejście T2EX (pin 2 procesora), ustawienie – uaktywnienie, wyzerowanie – dezaktywacja

**TR2 (bit T2CON.2, adres CAh)** – bit sterujący zliczaniem licznika T2, ustawienie (TR2=1) powoduje pracę licznika, wyzerowanie (TR2=0) zatrzymanie zliczania

**C/T2 (bit T2CON.1, adres C9h)** – bit określający funkcję pracy T2, i tak :  
– ustawienie (C/T2=1) przełącza T2 w tryb zliczania impulsów zewnętrznych  
– wyzerowanie (C/T2=0) włącza funkcję czasomierza (zliczanie impulsów wewnętrznych Fxtal/12)

**CP/RL2 (bit T2CON.0, adres C8h)** – bit określający tryb pracy licznika, i tak:  
– ustawienie (CP/RL2=1) powoduje aktywację trybu z zatraskiwaniem zawartości licznika (rejestrów T2 – TH2.TL2 w rejestrach RLD – RLDH.RLDL)  
– wyzerowanie (CP/RL2=0) powoduje pracę z automatycznym wpisywaniem wartości początkowej (z rejestrów RLD do rejestrów T2)

W zależności od kombinacji niektórych bitów z rejestru T2CON możliwe są różne stany pracy układu czasowo-licznikowego T2, oto kilka wskazówek:

- 1) W przypadku gdy ustawimy bit CP/RL2, licznik T2 będzie zliczał modulo  $2^{16}$ , po każdym przepełnieniu będzie ustawiany znacznik zgłoszenia przerwania TF2.  
Jeżeli ustawimy dodatkowo bit EXEN2, co spowoduje aktywację wejścia T2EX procesora (P1.1), to możemy zatrzasnąć podając na to wejście opadające zbocze, zawartość licznika T2 w rejestrach RLD (RLDH.RLDL).
- 2) Jeżeli bit CP/RL2 jest wyzerowany, licznik T2 będzie pracował w trybie z automatycznym wpisywaniem wartości początkowej z rejestrów RLD w momencie przepełnienia. W tym miejscu nasuwa się podobieństwo pracy liczników T0 lub T1 w trybie 1, z tym, że przeładowanie licznika następuje w przypadku T2 automatycznie, co zwalnia nas od programowego wpisywania wartości początkowej do rejestrów TH2 i TL2. W momencie przepełnienia ustawiany jest znacznik TF2, co może być także sygnałem zgłoszenia przerwania. Dodatkowo przeładowanie zawartości rejestrów roboczych licznika (TH2.TL2) może nastąpić pod wpływem zewnętrznego zbocza opadającego, podanego na wejście T2EX, trzeba tylko dodatkowo ustawić bit EXEN2 (EXEN2=1). Takie wymuszenie powoduje także ustawienie znacznika EXF2. Dzięki temu możliwe jest zsynchronizowanie pracy

wewnętrznego układu licznikowego T2 z zewnętrznym sygnałem zegarowym (dołączonym do wejścia T2EX).

- 3) W każdym trybie pracy ustawienie znacznika TF2 może być sygnałem zgłoszenia przerwania. W tym przypadku ustawiane znaczniki w słowie T2CON nie są automatycznie zerowane po przyjęciu przerwania, toteż należy o tym pamiętać w procedurze obsługi przerwania. Z drugiej strony niezzerowanie znaczników umożliwia programiście ich analizę w procedurze obsługi przerwania, a dopiero po tym ich wyzerowanie.
- 4) Jak wspominałem wcześniej, licznik T2 może taktować port szeregowy. W tym celu należy ustawić bity TCLK i RCLK. Taktowany w ten sposób port szeregowy będzie mógł pracować w trybie 1 (znaki 8-bitowe, prędkość określana programowo) oraz w trybie 3 (znaki 9-bitowe, prędkość określana programowo). Fizycznie w trybie taktowania portu licznika T2 pracuje zliczając impulsy z automatycznym ładowaniem wartości początkowej z rejestrów RLD (RLDH.RLDL). Zliczane mogą być impulsy wewnętrzne (czasomierz) o częstotliwości Fxtal/2, lub impulsy zewnętrzne z wejścia T2 (P1.0). Po podzieleniu impulsów taktujących (Fxtal/2 lub zewnętrznych z wejścia T2) dodatkowo przez 16 traktują one odbiornik (RCLK=1) lub nadajnik (TCLK=1) portu szeregowego. W tym trybie pracy licznika, jego przepełnienie nie ustawia znacznika jego przepełnienia TF2. Dzięki temu w przypadku gdy ustawimy bit EXEN2 (w słowie T2CON), to pod wpływem opadającego zbocza sygnału na wejściu T2EX (końcówka P1.1 procesora) ustawiony zostaje znacznik EXF2, co może być sygnałem zgłoszenia przerwania. Jak widać możliwe jest zatem wykorzystanie wejścia T2EX jako dodatkowego (obok INT0 i INT1) wejścia przerywającego procesora.

W przypadku taktowania portu szeregowego (T2 taktowany sygnałem wewnętrznym Fxtal / 2) prędkość transmisji (n) można określić wzorem:

$$n = \frac{F_{xtal}}{(65536 - RLD) \times 2 \times 16}$$

stąd łatwo po przekształceniu wzoru wyznaczyć wartość początkową rejestrów RLD przy zadanej prędkości transmisji:

$$RLD = 65536 - \frac{F_{xtal}}{2 \times 16 \times n} = 65536 - \frac{F_{xtal}}{32 \times n}$$

gdzie RLD to oczywiście wartość początkowa wpisana do rejestrów RLDH.RLDL. Poniżej podaję przykładowe prędkości transmisji dla rezonatora kwarcowego 11,0592 MHz.

| Fxtal (MHz) | RLD    | n (bodów)     |
|-------------|--------|---------------|
| 11,0592     | FFFFh  | 345600        |
| 11,0592     | FFFDh  | <b>115200</b> |
| 11,0592     | FFFCCh | 86400         |
| 11,0592     | FFFAh  | <b>57600</b>  |
| 11,0592     | FFF7h  | <b>38400</b>  |
| 11,0592     | FFF4h  | 28800         |
| 11,0592     | FFEEh  | <b>19200</b>  |
| 11,0592     | FFDCh  | <b>9600</b>   |
| 11,0592     | FFB8h  | <b>4800</b>   |
| 11,0592     | FF70h  | <b>2400</b>   |
| 11,0592     | FEE0h  | <b>1200</b>   |
| 11,0592     | FDC0h  | 600           |
| 11,0592     | FB80h  | 300           |
| 11,0592     | F700h  | 150           |
| 11,0592     | EE00h  | 75            |

Jak widać, zakres możliwych do uzyskania prędkości transmisji jest o wiele szerszy, niż w przypadku taktowania portu licznikiem T1. Najmniejsza szybkość w przypadku rezonatora kwarcowego jak w tabeli czyli 11,0592 MHz przy użyciu licznika T2, to 5 bitów na sekundę, a więc prawdziwy żółw! Sprawdźmy:  
 $n_{min} = 11059200 / (32 \times (65536 - 0)) = 11059200 / (32 \times 65536) = 5$  (bitów/sek.)

Wtedy oczywiście wartością początkową będzie zero (RLDH.RLDL=0). W tabeli pogrubioną czcionką zaznaczono typowe, spotykane w komputerach PC wartości transmisji szeregowej, realizowanej poprzez port RS232c.

A oto przykład programowania rejestru sterującego T2CON oraz roboczych i dodatkowych w celu uzyskania kilku trybów i funkcji pracy licznika T2.

Załóżmy, że nasz licznik T2 będzie pracował jako czasomierz z automatycznym ładowaniem wartości początkowej z RLD po przepełnieniu. Założymy, że przepełnienie ma następować dokładnie co 1ms, a do procesora dołączony jest rezonator kwarcowy o częstotliwości 6 MHz. Obliczenia:

- przy fxtal= 6 MHz licznik pracując jako czasomierz będzie inkrementowany co 2µs

## Też to potrafisz

– w czasie 1 ms (milisekundy) zawiera się 1000  $\mu$ s (mikrosekund), czyli 500 okresów zegara procesora  
– wobec tego wartość początkowa licznika można obliczyć jako:  
 $\text{wart.początkowa} = \text{wartość maksymalna} - 500 = 65536 - 500 = 65036 = \text{FE0Ch}$  (heksadecymalnie)  
można więc zapisać komendy inicjujące licznik T2:  
`mov T2CON, #0 ;zasomierz z automat.`  
`;ładowaniem wart. początkowej z RLD`  
  
`mov RLDH, #0FEh`  
`mov RLDL, #0Ch ;załadowanie wartości FE0Ch`  
`;:(początkowej)`  
  
`mov TH2, RLDH`  
`mov TL2, RLDL ;aby prawidłowo zainicjować`  
`;pierwsze przepełnienie`  
`setb TR2 ;start licznika T2`  
  
`.....`  
`..... ;dalsze instrukcje`

Analizując linie poleceń warto szczególną uwagę zwrócić na pierwszą komendę, która ustawia bity w słowie T2CON. Zauważmy, że wszystkie bity tego słowa zostały ustawione na zero, co zgadza się z opisem rejestru T2CON, który omówiłem przed chwilą.

Jeżeli ktoś teraz zechce dopisać procedurę obsługi przerwania, może to zrobić analogicznie jak w przypadku opisanej wcześniej procedury obsługi przerwania od przepełnienia licznika T0 (T1), pamiętając jednak o adresie wektora przerwania, który w tym przypadku wynosi:

**002Bh**

a w przypadku naszego komputerka edukacyjnego (AVT-2250) procedura powinna zaczynać się od adresu

**802Bh**

zgodnie z zasadami pisania takich podprogramów, które omówiłem w poprzednim numerze EdW.

Program na nasz komputer mógłby więc zaczynać się następująco:

```
org 8000h
ljmp START
org 802Bh

intT2:
.....
.....
pop DPL
pop DPH
pop Acc
reti
;*****
START:
mov T2CON, #0 ;zasomierz z automat.
;ładowaniem wart.
początkowej z RLD

mov RLDH, #0FEh
mov RLDL, #0Ch ;załadowanie wartości FE0Ch
;:(początkowej)

mov TH2, RLDH
mov TL2, RLDL ;aby prawidłowo zainicjować
;pierwsze przepełnienie

setb TR2 ;start licznika T2

.....
..... ;dalsze instrukcje
.....

END
```

Nie należy jednak zapominać że w licznik T2 wyposażony jest procesor 8052/C, toteż jeżeli masz w komputerku AVT-2250 procesor 8051/C51, to musisz go po prostu wymienić na właściwą kostkę. W sklepie nie powinna ona kosztować więcej jak 5zł.

## Specjalne tryby pracy

W tej części artykułu omówię tryby pracy procesora, dzięki którym możliwe jest realizowanie ciekawych rozwiązań układowych, np. urządzeń zasilanych z baterii – czyli takich, w których kwestia poboru energii jest elementem krytycznym.

W obszarze rejestrów SFR procesora (przypominam – jest to wewnętrzna pamięć danych RAM adresowana bezpośrednio, o adresach 80h...FFh) znajduje się jeszcze jeden ciekawy rejestr specjalnego przeznaczenia. Jego funkcją jest kontrola specjalnych trybów pracy procesora, a mianowicie:

- trybu tzw. „jałowego”,
- trybu tzw. „uśpienia”.

Z grubsza rzecz ujmując tryby te różnią się od siebie stopniem poboru mocy przez procesor, oraz funkcji, jakie pozostają aktywne w tych trybach pracy w odróżnieniu od normalnego trybu pracy procesora.

Przejdźmy zatem do omówienia rejestru PCON, bo o nim jest mowa. Poniżej przedstawione jest znaczenie poszczególnych bitów tego rejestru. Warto przy tym zauważyć, że rejestr nie może być adresowany bitowo, toteż nie da się sterować jego poszczególnymi bitami poprzez instrukcje typu:

`SETB bit`  
lub  
`CLR bit`

|        |      |   |   |   |     |     |    |     |  |        |
|--------|------|---|---|---|-----|-----|----|-----|--|--------|
| nazwa: |      |   |   |   |     |     |    |     |  | adres: |
| PCON   | SMOD | - | - | - | GF1 | GF0 | PD | IDL |  | 87h    |

Rejestr PCON jest umieszczony pod adresem 87h w obszarze SFR procesora. Zawiera 5 istotnych dla użytkownika bitów.

**SMOD (bit .7)** – bit podwojenia szybkości transmisji poprzez port szeregowy w trybach 1,2 lub 3 pracy. Ustawienie tego bitu (SMOD=1) powoduje dwukrotne zwiększenie częstotliwości taktowania portu szeregowego poprzez licznik T1, kiedy ten pracuje w trybie taktowania tego portu. W odcinku, w którym omawiałem port szeregowy, w zamieszczonej tam tabeli podałem przykłady wartości początkowych dla standardowych prędkości transmisji asynchronicznej. Wszystkie odnoszą się właśnie do podwojonego trybu prędkości transmisji, kiedy bit ten jest ustawiony. Jeżeli nie chcemy pracować w trybie podwojonej prędkości, bit ten powinien być wyzerowany (PCON=0).

Ustawienie bitu SMOD w rejestrze PCON można wykonać za pomocą instrukcji, np.:

`ORL PCON, #80h`

wyzerowanie zaś za pomocą instrukcji:

`ANL PCON, #7Fh`

– (bity: 6...4) – nie wykorzystane

**GF1 (bit .3)** – bit programowy do dowolnego wykorzystania przez programistę

**GF0 (bit .2)** – bit programowy do dowolnego wykorzystania przez programistę

**PD (bit .1)** – bit włączający tryb obniżonego poboru mocy – „uśpienia”. Ustawienie tego bitu powoduje wprowadzenie procesora w tryb uśpienia, kiedy to pobór prądu spada o około 500 razy, a napięcie zasilające Vcc może być obniżone do 2,0V. Wykonanie instrukcji ustawiającej ten bit jest ostatnim poleceniem wykonanym przez procesor w programie.

**IDL (bit .0)** – bit włączający tryb „jałowy” procesora.

Poniżej nieco dokładniej omówię oba tryby pracy.

## Tryb jałowy

Instrukcja, która ustawia bit PCON.0 powoduje wprowadzenie procesora w ten tryb. Jest ona ostatnią wykonywaną przez procesor instrukcją. Wewnętrzny sygnał zegarowy zostaje odłączony od jednostki centralnej (CPU), ale układ przerwań, port szeregowy i licznikowy pracują dalej, jeżeli wcześniej były odpowiednio skonfigurowane i ustawione. Stan całego procesora, a więc stan:

- rejestrów specjalnych SFR,
  - pamięci wewnętrznej RAM użytkownika
  - pinów portów P0...P3
- pozostaje bez zmian i jest taki sam jak był tuż przed wejściem procesora w tryb jałowy.

Końcówki ALE i /PSEN procesora ustawiają się w stan wysoki.

Istnieją dwa sposoby na wyjście z tego stanu:

- 1) Nadejście dowolnego przerwania – oczywiście jeżeli było ono wcześniej uaktywnione w rejestrze IE. Pojawienie się przerwania zeruje automatycznie (bez udziału programu użytkownika) flagę PCON.0 – i procesor powraca do normalnej pracy, z tym, że następną instrukcją po wyjściu ze stanu jałowego pod wpływem przerwania będzie pierwsza znajdująca się w procedurze obsługi danego przerwania, aż do instrukcji RETI, kiedy to procesor automatycznie powraca do instrukcji następnej po tej, która wprowadziła procesor w stan jałowy, czyli tej, która ustawiła bit IDL w rejestrze PCON.
- 2) Drugim sposobem na wyjście z tego stanu jest zerowanie procesora. Ze względu na fakt, że podczas trybu „jałowego” procesor nadal zegar systemowy, do prawidłowego zresetowania potrzebny jest impuls zerujący o długości co najmniej 24 okresów oscylatora.

## Tryb uśpienia – obniżonego poboru mocy

W tym trybie, obecnie stosowanym przez producentów tylko w kosztach typu CMOS, czyli np. 80C51 lub 80C, cały mikrokontroler pobiera znacznie mniej energii, oraz dodatkowo napięcie zasilające układ może zostać zmniejszone od standardowych 5V do 2,0V. Instrukcja ustawiająca bit PD (PCON.1) jest ostatnią wykonywaną przez procesor. W trybie tym oscylator procesora

zostaje wyłączony (po prostu staje). Zostają odłączone wszystkie układy funkcjonalne procesora, takie jak układy licznikowe, port szeregowy, układ przerwań. Pozostaje jedynie niezmienniona zawartość wewnętrznej pamięci RAM, w tym pamięci użytkownika oraz rejestrów specjalnych SFR. Piny portów pozostają zgodne ze stanami odpowiadających im bitów w rejestrach P0...P3 w obszarze SFR. Końcówki ALE i PSEN znajdują się w stanie niskim. W tym trybie pracy procesora, a raczej nie trybie pracy, co uśpienia, procesor pobiera około 500 razy mniej prądu niż w stanie normalnej pracy. Dla przykładu podam, że dla kostki 80C51 (czyli w wersji CMOS) :

- w trybie pracy normalnej pobór prądu wynosi ok. 20mA (przy  $f_{xtal}=12\text{MHz}$ )
- w trybie „jałowym” pobór prądu spada do około 3,0 mA (przy  $f_{xtal}=12\text{MHz}$ )
- w trybie uśpienia pobór prądu przez układ wynosi ok. 50µA (mikroamper!), przy obniżeniu zasilania do 2,0V.

Jedyną metodą na opuszczenie trybu uśpienia i powrót do normalnej pracy jest wyzerowanie mikroprocesora poprzez podanie impulsu resetującego na wejście RST (pin 9) o czasie trwania ok. 10ms.

## Reset procesora

Popularne zresetowanie odbywa się poprzez podanie impulsu dodatniego na wejście RST kostki (pin 9) zgodnie z zasadami, które omówiłem przed chwilą. Najprostsze i bardziej rozbudowane układy resetowania mikrokontrolerów serii MCS-51 podał w jednym z pierwszych odcinków klasy mikroprocesorowej.

W wyniku zresetowania rejestru układy specjalne procesora (SFR) zostają automatycznie zainicjowane wartościami, jak podaję w tabeli poniżej:

| Rejestr | Wartość po „RESET” | Uwagi                |
|---------|--------------------|----------------------|
| PC      | 0000h              | licznik rozkazów     |
| ACC     | 00h                | akumulator           |
| B       | 00h                | rejestr B            |
| PSW     | 00h                | słowo stanu programu |
| SP      | 07h                | wskaźnik stosu       |

|             |           |                                 |
|-------------|-----------|---------------------------------|
| DPTR        | 0000h     | wskaźnik danych                 |
| P0...P3     | FFh       | porty                           |
| IP          | xxx00000b | rejestr priorytetu przerwań     |
| IE          | 0xx00000b | rejestr masek przerwań          |
| TMOD        | 00h       | rej. liczników T0 i T1          |
| TCON        | 00h       | rej. ster. liczników i przerwań |
| TH0         | 00h       |                                 |
| TL0         | 00h       |                                 |
| TH1         | 00h       |                                 |
| TL1         | 00h       |                                 |
| SCON        | 00h       | rejestr portu szeregowego       |
| SBUF        | zmienny   | bufor portu szeregowego         |
| PCON (MOS)  | 0xxxxxxx  | dla układów NMOS (8051/2)       |
| PCON (CMOS) | 0xxx0000b | dla ukl. CMOS (80C51/52)        |

Uwaga: litera „x” oznacza, że dany bit przyjmuje wartość przypadkową lub nie jest implementowany w danym rejestrze. Wartość po „RESECE” przedstawiono w formacie szesnastkowym lub binarnym (z „b” na końcu) celem ułatwienia analizy i porównania z opisem rejestrów SFR na wkładce z numeru Edw 11/97.

Programując mikrokontrolery 8051/52, trzeba pamiętać o tym fakcie, toteż szczególnie w przypadku wykorzystania specjalnych trybów pracy należy mieć na względzie fakt, że rejestry specjalne zostają utracone po resetie procesora, a więc konieczne jest ich odtworzenie, oczywiście tylko w razie takiej konieczności.

Jeżeli ktoś interesuje się szczegółami, dotyczącymi architektury wewnętrznej mikrokontrolera 80C51 lub podobnych, które nie za często są wykorzystywane w projektach, przynajmniej przez początkujących programistów, może sięgnąć do literatury [1] i [2]. Obie pozycje wymagają jednak znajomości podstaw terminologii technicznej jęz. angielskiego. Chętnych zapraszam do lektury, pozostałym proponuję poczekać na kolejny odcinek naszego cyklu.

Sławomir Surowiński

Literatura:

- [1] 80C51-based 8-bit Microcontrollers, Data Handbook, Philips 1995  
[2] Microcontroller Data Book, Atmel 1995,6,7

# Lekcja 9

Na początku lekcji zajmijmy się rozwiązaniem zadań z poprzedniego numeru EdW i odcinka klasy mikroprocesorowej.

## Rozwiązanie zadania nr 1

Oto krótka analiza głównej części listingu od etykiety START.

W linii (72) profilaktycznie zatrzymujemy licznik T1. W dwóch kolejnych liniach ustawiamy tryb pracy licznika T1, jako czasomierza zliczającego wewnętrzne impulsy zegarowe (tryb pracy 0). W linii (75) wpisujemy obliczoną wartość początkową, przy której licznik będzie przepelniany okładnie co 1/128 sekundy.

W linii (76) wyzerowany zostaje licznik zliczający 1/128 części sekundy, a w kolejnej linii (77) załadowany zostaje do zmiennej systemowej komputerka starszy bajt 16-bitowego adresu tabeli wektorów przerwań w zewnętrznej pamięci programu.

Należy jeszcze w linii (78) odblokować przerwanie od licznika T1 (przepelnienie licznika) oraz ustawić priorytet na to przerwanie (linia 79) inaczej bowiem odmierzenie czasu może być zakłócone pracującym przecież stale licznikiem T0 i wywołaną przez jego przepelnienia procedurą obsługi przerwania, która zajmuje się obsługą wyświetlacza i klawiatury komputerka edukacyjnego AVT-2250.

Po zainicjowaniu układu licznikowego T1 oraz przerwania od tego licznika, w linii 81 profilaktycznie czyścimy wyświetlacz, aby potem w liniach (82)...(85) pobrać z klawiatury godzinę początkową i w linii 86 zapamiętać ją w komórce GODZ. Podobnie dzieje się dla minut – linie (88)...(91) i sekund – linie (94)...(98).

Po wprowadzeniu (ustawieniu czasu) zapalone zostają poziome kreski oddzielające godziny od minut (linia 100) i minuty od sekund (linia 101), a następnie po koniecznym ze względu na drgania styków klawiatury opóźnieniu (ok. 0,5 s.) – linie (102),(103), komputer czeka na naciśnięcie klawisza przez użytkownika celem uruchomienia zegara. Uruchomienie następuje w kolejnej linii (105), a dalej znajduje się część programu, której zadaniem jest wyświetlanie upływającego czasu na wyświetlaczu – od etykiety „pokaz:” w linii (107).

Dzięki badaniu bitu nr 6 w zmiennej „licz128”, który przecież zmienia swój stan co 0,5 sekundy (linia 109), możliwe jest naprzemienne gaszenie kresek (linie 110,111) oraz ich zapalanie – linie (113) i (114).

Dalej od etykiety „czas:” rozpoczyna się wypisanie czasu, czyli wyświetlane są godziny – linie (116...118), minuty – linie (119...121) oraz sekundy – linie (122...124).

W linii (125) następuje skok do początku, gdzie następuje kolejne uaktualnienie wyświetlanego czasu.

## Rozwiązanie zadania nr 2

Aby wyświetlić aktualny czas w trybie jak pokazano w zadaniu, czyli:

1 2 – 3 4 .5 8

wystarczy zmodyfikować podane linie na postać jak poniżej:

| linia | adres | kod    | instrukcja      |
|-------|-------|--------|-----------------|
| 101   | 8099  | 757D80 | mov DL6,#kropka |
| 111   | 80AE  | 757D80 | mov DL6,#kropka |
| 114   | 80B6  | 757D80 | mov DL6,#kropka |

lub w zależności od swoich upodobań posłużyć się zmienną systemową: „blinks”, której każdy z bitów określa atrybut wyświetlanego znaku, czy ma migać, czy być normalnie wyświetlanym.

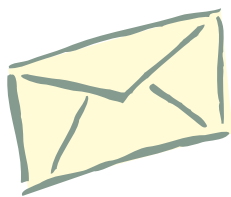
Ponieważ nie dotarli do mnie jeszcze listy z rozwiązaniami zadania nr 3, pozwolę sobie zamieścić najciekawsze propozycje w kąciку pocztowym 8051 w kolejnym numerze EdW.

## Uwaga!

W poprzedniej lekcji nr 8 do listingu wkładły się drobne błędy, a mianowicie, komentarze w liniach o podanych numerach powinny wyglądać następująco:

- (74) ..... ;T1 jako 8-bitowy z preskalarem 5-bitowym  
(76) ..... ;wyzerowanie licznika 1/128 s.

Sławomir Surowiński



## Kącik pocztowy 8051

Zasypany wieloma listami, dotyczącymi cyklu artykułów poświęconych programowaniu mikrokontrolerów 8051, postanowiłem uruchomić kącik pytań i odpowiedzi, które kierujecie do mnie w swoich listach.

Na wstępie chciałem bardzo podziękować za każdy list, zarówno te pochwalne jak i krytyczne. Przysnam że, cieszy mnie bardzo fakt, iż tak wielu z Was zdecydowało się sięgnąć po „mikroprocesory”, a co najważniejsze odnosi już małe, ale jak ważne w nauce sukcesy.

Dzisiaj kolejna porcja listów od Czytelników i odpowiedzi na niektóre pytania.

### List 1

**Zenon Rakoczy z Chropaczowa** ma wątpliwości co do instrukcji

`MOV adres1, adres2`

a ponieważ, jak pisze, jest „ręcznikiem” problem pojawił się w transkrypcji na języka maszynowy bez korzystania z komputera i assemblera 8051.

Jak pisałem wcześniej w artykułach instrukcje typu

`MOV X, Y`

kopiuje zawartość po stronie Y do literału X, czyli następuje przeniesienie typu

`X ← Y`

i wszystko się zgadza. Zenon ma wątpliwości, jak należy przetłumaczyć np. instrukcję:

`MOV DPH, B`

Jak wynika z wcześniejszych analiz instrukcji tego typu, najpierw należy zapisać bajt określający instrukcję MOV tego typu, czyli zgodnie z tabelą we wkładce będzie to:

85 – kod instrukcji „MOV adres1, adres2”

83 – adres rejestru DPH

F0 – adres rejestru B, czyli

instrukcję tę można zapisać jako ciąg bajtów: 85 83 F0.

Niestety, jak podawałem wcześniej, podczas opisu instrukcji procesora ten typ instrukcji jest wyjątkiem i kolejność rejestrów po przetłumaczeniu będzie odwrotna, czyli: **85 F0 83**.

Oczywiście nie zmienia to działania tej instrukcji, po prostu tak tłumaczy się ją na język maszynowy.

Przy okazji podczas omawiania tej instrukcji w EdW w opisie wkładki się błąd, było zatem:

`MOV adres1, adres2`

– przepisanie zawartości komórki o adresie „adres1” do komórki o adresie „adres2”

a powinno być:

`MOV adres1, adres2`

– przepisanie zawartości komórki o adresie „adres2” do komórki o adresie „adres1”

dalsza część opisu jest bez błędów, czyli:

(adres1) ← (adres2)

- kod: 1 0 0 0 1 0 1

- cykl: 2 bajt: 3 (kod instrukcji + adres2 + adres1)

- przykład:

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

### List 2

**Marcin Jurzak** nadesłał wiadomość przez Internet, że ma kłopoty z przysyłaniem programów z komputera do komputerka. Za każdym razem, kiedy transmisja zaczyna się na wyświetlaczu komputerka edukacyjnego, pojawia się napis „Err” – czyli komunikat błędu.

Jeżeli taki komunikat pojawia się oznacza to, że kabel wykonał poprawnie, i dane przesyłane są z PC-ta do komputerka AVT-2250, z tym, że nie są zrozumiałe dla BIOS-a, stąd komunikat o błędzie. Powodów takiego stanu rzeczy może być kilka:

a) nie ustawione parametry portu szeregowego w PC-cie – powinien wydać komendę ustawiającą je, jak opisywałem przy okazji opisu Bios-a komputerka, a mianowicie z poziomu DOS-a wywołać komendę:

`MODE COM2: 4800, n, 8, 1 {Enter}`

Jeżeli korzysta z Windows 95, należy parametry portu ustawić w Panelu sterowania w opcji System – Menedżer Urządzeń, ustawiając parametry:

Bitów na sekundę : 4800

Bitów danych: 8

Parzystość: brak

Bitów stopu: 1

Sterowanie przepływem: Brak

oraz dodatkowo w opcji „Zaawansowane...” konieczne wyłączyć (odhaczyć) opcję buforowania poprzez FIFO.

b) inną przyczyną może być fakt, że próbuje wysłać zbiory binarne, a nasz komputer akceptuje tylko zbiory w formacie Intel-HEX, rzadziej sprawdź.

Czytelnik ma także wątpliwości co do rysunku kabla połączeniowego PC z komputerkiem AVT-2250 w wersji 9 na 25 pinów. Informuję, że rysunek jest prawidłowy, a skrzyżowanie 2 z 3 występuje tylko w kablu „9 na 9”. We wtyku DB25 znaczenie końcówek 2 i 3 jest dokładnie odwrotne niż w końcówce DB9, stąd brak krzyżowych połączeń.

### List 3

Łyżka miodu od Czytelników: **Wiesław Kusek z Mielca** pisze:

„...Otrzymałem niedawno obiecany bezpłatnie zestaw AVT-2250. Od razu chcę bardzo podziękować za ten zestaw. I w związku z tym mam kilka uwag i spostrzeżeń, oraz co nie udało mi się uniknąć – pytań. ... Kit AVT-2250 – bardzo dobra ocena. Rzecz droga, ale wiadomo pewnych rzeczy nie da się przeskoczyć. Układ przemysłowy i dopracowany do końca, co świadczy o dużej fachowości autora opracowania i co dla mnie – początkującego w tym temacie najważniejsze – w dość jasny i czytelny sposób opisany. Chociaż kilku zdań nie rozumiem. Brawo za wysoką jakość płytek drukowanych. Układ został przeze mnie zmontowany i po włączeniu zasilania ruszył bez żadnego problemu.”

**Adam Szendzielorz z Wodzisławia Śląskiego** napisał do nas:

„... Niedawno otrzymałem od Was w/w układ „mikrokomputerka edukacyjnego.... Po odebraniu go na pocztę, czym prędzej wziąłem się do montowania go. W jakieś 3 godziny układ stał na stole gotowy – trzeba było tylko jeszcze sprawdzić czy działa... Po podłączeniu zasilania na wyświetlaczach ukazał się napis „HELLO” !!! Sam nie mogłem uwierzyć – po raz pierwszy w mojej karierze już po pierwszym uruchomieniu (w sumie dość skomplikowanego układu) działał on poprawnie!!! – to chyba dzięki bardzo starannie wykonanych płytek drukowanych i sprawnych elementów, choć moja staranność przy jego wykonywaniu też na pewno temu sprzyjała. (”Ależ oczywiście! – przyp. red.) Po pierwszych emocjach zacząłem przerabiać wszystkie lekcje EdW dotyczące procesorów – nie jest to łatwe, ale myślę, że w drodze praktyki będzie to prostsze i lepiej to zrozumieć.

Ponieważ posiadam komputer PC, wykonałem kabel łączący go z układem. I tu mała uwaga i jednocześnie prośba... Ręczne wklepywanie to chyba strata czasu (który można by wykorzystać do innych celów), pisanie na komputerze to już coś – można to zapisać, ponownie odtworzyć i przesłać do komputerka, łatwiej coś zmienić, a przede wszystkim wykonuje się to „duuuużo” szybciej.”

Takich listów otrzymuję bardzo dużo. Cieszymy się z tego, że większość z Was nie ma problemów z uruchamianiem układu komputerka edukacyjnego AVT-2250. Jednak do redakcyjnego serwisu trafiają czasem z reklamacją zestawy nie działające. Powodem takiego stanu rzeczy i często zażenowania nabywcy jest niestaranność, często budzący zgrozę sposób montażu. Apeluję więc, nie starajcie się lutować elementów „byle czym” i mierzcie swoje siły na zamiary, a każdy układ elektroniczny odpali bez problemów. Jeżeli nie czujecie się na siłach w samodzielnym zmontowaniu, proponuję zamówienie w AVT zmontowanego komputerka – kit AVT-2250/C.

P.S. Adamie z Wodzisławia, brakującą dyskietkę AVT-2250/D otrzymaj jak tylko pojawią się one w magazynie AVT. W razie kłopotów proszę o kontakt z Działem Handlowym AVT i potwierdzenie swego zamówienia.



## List 4

Tomasz Jabłonowski z Moniek pisze:

- że, po pierwsze, nie może znaleźć „we wkładce” (zapewne EdWV) np. CLS, A2HEX itp.
- po drugie, „inne” kompilatory nie rozumieją argumentów niektórych publikowanych w cyklu komend, np.  
DL1,#\_minus

Szanowny kolego, wyrażenia typu CLS, A2HEX, DPTR4HEX, to nie są komendy procesora 8051, ale zaproponowane przez autora cyklu szkoły mikroprocesorowej nazwy procedur (podprogramów) umieszczonych w pamięci EPROM komputerka. Pisałem o tym wiele razy. Pod nazwami tymi kryją się konkretne adresy, proszę zajrzeć do zbiorów BIOS.INC, które znajdują się na dyskietce AVT-2250/D.

To samo dotyczy niektórych zdefiniowanych konkretnie dla naszego komputerka edukacyjnego stałych i zmiennych, a więc:

**DL1** – pod tą nazwą kryje się adres komórki w wewn. RAM procesora, która przechowuje znak do wyświetlenia na pozycji nr 1 (pierwszy wyświetlacz).

**\_minus** – pod tą nazwą kryje się stała (liczba), która określa kolejność zapalonych segmentów wyświetlacza podczas wyświetlania znaku „-”.

Toteż jeżeli zechcesz używać innego kompilatora, powinienes zadeklarować na początku programu przed instrukcjami następujące przypisania:

```
DL1 equ 78h
_minus equ 01000000b
```

To samo dotyczy wszelkich innych nazw, którymi operujemy w naszych artykułach, a które można znaleźć w zbiorach CONST.INC i BIOS.INC na dyskietce do systemu AVT-2250.

## List 5

Bardzo ciekawy list od Czytelnika z Katowic **Marek Joniec** pyta mianowicie:

- 1) „... dlaczego wszelkie napisy adresów i danych dokonuję w kodzie heksadecymalnym, skoro procesor i układy zewnętrzne operują danymi binarnymi, w którym momencie i gdzie np. dana „D9h” jest transkodowana na liczbę dwójkową „11011001”.
- 2) Czy jeżeli system działa tylko z zewnętrzną pamięcią RAM, która w naszym przypadku posiada przestrzeń adresową 8000h...FFFFh, to gdzie mam w niej szukać obszaru rejestrów specjalnych SFR? Czy obszar ten jest automatycznie gdzieś umiejscowiony, czy też odpowiedzialny jest za to monitor systemowy?
- 3) Jeżeli w specyfikacji rejestrów podane jest, że rejestr PSW posiada adres D0h, to gdzie się on znajduje w moim RAM-ie?
- 4) Dlaczego chcąc wpisać jakąś liczbę na dany wyświetlacz odwołuję się do adresów 78h...7Fh (przecież są to adresy pamięci programu EPROM, a nie pamięci RAM).
- 5) Większość rozkazów asemblerowych wymaga podania adresu bezpośredniego „xx”, co to znaczy i jak mam adresować obszar 8000h...FFFFh za pomocą dwóch pozycji?
- 6) W jaki sposób mam wprowadzać dane z klawiatury do pamięci nie używając gotowego polecenia GETACC zawartego w monitorze?
- 7) Na czym polega w praktyce adresowanie bitowe rejestrów specjalnych?

## Oto odpowiedzi

**Ad.1** Zapisywanie wszelkich wartości liczbowych oraz kodu maszynowego procesora w zapisie szesnastkowym (heksadecymalnym) jest najbardziej naturalnym i czytelnym, jak się wkrótce przekonasz, sposobem. Szesnastkowy zapis, szczególnie w ujęciu techniki cyfrowej, mikroelektroniki, mikrokontrolerów i wreszcie komputerów jest tym, czym język narodowy danego kraju.

Prawdą jest, że układy komputerowe przetwarzają wszystko w kodzie binarnym, ale nam ludziom trudno byłoby „czytać” i analizować kod programu w takiej postaci, aczkolwiek jeszcze przed kilkudziesięciami laty taki sposób programowania był jedynym – lecz były to lata 50. i początki techniki komputerowej.

Prosty przykład, łatwiej odnaleźć (wzrokowo) i rozróżnić dwie liczby, np.:

```
1234h i ABCDh
niż te same zapisane w postaci binarnej, a więc:
0001001000110100b i 1010101111001101b
```

Jeżeli chodzi o drugą, dość oryginalną część pytania, to mogę mieć pewien problem z odpowiedzią, bowiem fakt zamiany, o której Czytelnik mówi, można by porównać z czymś tak oczywistym, a jednocześnie trudnym do opisania jak np. „słuchanie czyjejs mowy i wyciąganie z niej wniosków”.

Aby jednak zaspokoić ciekawość Czytelnika można powiedzieć, że w przypadku mikrokontrolerów zamiana liczby z kodu szesnastkowego na binarny fizycznie odbywa się w momencie:

- w przypadku pobierania rozkazów z pamięci EPROM komputerka – w momencie programowania pamięci EPROM w urządzeniu zewnętrznym zwanym programatorem pamięci EPROM, który wczytuje zbiory np. Intel-HEX (takie jakie tworzy nasz kompilator) a następnie zapisuje dane o programie w pamięci EPROM korzystając z linii adresowych kostki pamięci oraz z linii (np. 8) danych D0...D7, których przecież jest 8 tak jak jest 8 bitów w jednym bajcie informacji.
- w przypadku przesyłania danych z komputera PC do komputerka – już w momencie transmitowania danych z komputera, który zamienia informację na szeregową, czyli ciąg bitów z odpowiednimi sygnałami sterującymi.

Zresztą najlepszą odpowiedzią na to pytanie może być fakt, że sam mikrokontroler 8051 i wszystkie pozostałe układy tego typu pobierają i wysyłają informację już w postaci binarnej poprzez swoje końcówki w obudowie, których może być więcej lub mniej. W przypadku 8051 i podobnych mu są to 4-8-bitowe porty P0...P3.

Tak więc informacja trafia do procesora – wychodzi z niego zawsze w postaci binarnej, to tylko peryferyjne układy zewnętrzne transformują ją na postać bardziej czytelną dla użytkownika czy programisty i o to przecież chodzi, tak działa postęp.

Dlatego w komputerze PC używamy monitora czy klawiatury, które przecież też za pomocą dodatkowych urządzeń (takich jak karta graficzna) zamieniają informację bitową na postać bardziej czytelną np. szesnastkową i wyświetlają ją na ekranie, czy zamieniają wciśnięty klawisz na sekwencję bitów odpowiadającą kodowi danego klawisza.

**Ad.2** Jeszcze raz wyjaśniam.

- a) Istnieją dwa rodzaje pamięci danych – **wewnętrzna i zewnętrzna**
- b) **wewnętrzna zawarta jest fizycznie w strukturze procesora 8051** (w kostce).

Procesor ten zawiera dokładnie: 128 komórek (bajtów) pamięci użytkownika o adresach: 00h...7Fh oraz 128 komórek (nie wszystkie aktywne) o adresach 80h...FFh, pod którymi to znajdują się rejestry specjalne SFR procesora.

**To wszystko zawarte jest w strukturze procesora, pamiętajmy!**

Adresowanie tej pamięci RAM (wewnętrznej) odbywa się za pomocą wszystkich rozkazów procesora typu MOV oraz rozkazów wykonujących operacje arytmetyczne lub logiczne na komórkach tej pamięci, czyli: ANL, ORL, XRL, ADD, SUBB, INC, DEC, itd.

- c) zewnętrzna pamięć RAM to pamięć, jak sama nazwa wskazuje, dołączana z zewnątrz w postaci kostek SRAM o rozmiarze maksymalnie 64kB (65536 bajtów). Adresowanie tej pamięci odbywa się za pomocą tylko 2 rozkazów, a mianowicie:

**MOVX A, @DPTR** – odczyt danej z komórki w **zewnętrznej** pamięci danych o adresie zawartym w DPTR (czyli pokrywającym całe 64kB, 0...FFFFh) i umieszczenie jej w akumulatorze (Acc), czyli rejestrze w **wewnętrznej** RAM procesora o adresie bezpośrednim E0h.

**MOVX @DPTR, A** – zapis danej z komórki z akumulatora do **wewnętrznej** pamięci danych pod adres wskazany w rejestrze DPTR (uwaga! jw.).

**Ad.3** Po odpowiedzi na pyt. poprzednie odpowiedź na to pytanie jest oczywista – w PSW znajduje się w wewnętrznej pamięci RAM procesora (w kostce) pod adresem D0h.

**Ad.4** To nieprawda, że adresy „odwołań do wyświetlacza” znajdują się w EPROM-ie. Pisząc program wykorzystujący procedury BIOS-a komputerka edukacyjnego AVT-2250 programista może dla ułatwienia skorzystać z usług prowadzonych przez monitor.

I tak, fizycznie monitor komputerka w procedurze obsługi przerwania (pojawiającej się okresowo dokładnie 512 razy na sekundę) pobiera cyklicznie zawartość kolejnych rejestrów DL1...DL8, czyli o adresach 78h...7Fh w **wewnętrznej** RAM procesora, a następnie przepisuje je do zatrasku (74574) na płycie wyświetlacza, który to steruje układem ULN2803A – patrz schemat komputerka edukacyjnego. Sekwencję tę można zilustrować następująco:

```
MOV A, DL1 ;załadowanie do akumulatora
 ;zawartości rejestru DL1
```

```
MOV DPTR, #4000h ;załadowanie do DPTR adresu rejestru 74574
MOVX @DPTR, A ;wreszcie przesłanie zawartości akumulatora
 ;do rejestru celem wyświetlenia
```

**Ad.5** Po przeczytaniu odpowiedzi na pytanie 2, wiesz już, drogi Czytelniku, że nie da się zaadresować zewnętrznej pamięci RAM

## Też to potrafisz

(0000...FFFFh) za pomocą rozkazów z argumentami adresu jako „xx”, a jedynie rozkazami typu MOVX.

Ad.6 Wprowadzanie danych do pamięci komputerka bez użycia procedury GETACC jest oczywiście wykonalne. Trzeba tylko napisać krótki programik, który będzie realizował tę funkcję, poniżej podaję kod źródłowy polecenia, który odpowiada temu zawartemu w BIOS-ie komputerka.

\*\*\*\*\*  
GETACC:

```
push B
acall GETDIGIT;pobiera Acc z klawiatury z poz w B
swap A
push Acc
inc B
acall GETDIGIT
pop B
add A,B
pop B
ret
```

\*\*\*\*\*  
;Procedura pomocnicza pobierająca znak z klawiatury z wyświetleniem  
GETDIGIT: ;pobiera cyfrę z pozycji w B

```
push DPH
push DPL
push B
mov A,#DL1
add A,B
dec A
mov R0,A ;adres pozycji displeja
acall CONIN
cjne A,#klaw_OK,nieok
sjmp czek1
clr C
nieok: subb A,#30h
mov DPTR,#kodyk
movc A,@A+DPTR ;pobranie wartości bajtu
(0...15)
mov B,A
mov DPTR,#cyfry
movc A,@A+DPTR ;pobranie znaku na displej
mov @R0,A ;i wyświetlenie go
mov A,#nullkey
acall DELAY
mov A,B
pop B
pop DPL
pop DPH
ret
```

Zapis danej w pamięci komputerka pod wskazanym adresem można zrealizować także za pomocą takiego prostego listingu:

\*\*\*\*\*  
EXRAM\_ZAPIS:

```
mov DPTR,#adres ;w miejsce „adres” podać adres
mov A,#dana ;w miejsce „dana” wpisać daną
movx @DPTR,A ;zapis do zewn. RAM
ret
```

Problem tylko w tym, że aby zapisać np. 100 komórek pamięci, trzeba za każdym razem podawać inny adres i daną. I po to są takie procedury standardowe jak GETACC, CONIN, aby to robić bezboleśnie i bez potrzeby kompilowania programiku EXRAM\_ZAPIS np. 100 razy.

Ad.7 Na koniec bardzo słuszne pytanie na temat mało poruszany w naszym cyklu a mianowicie **adresowania bitowego rejestrów**.

Adresowanie bitowe rejestrów polega na odczycie lub modyfikowaniu (zapisie) poszczególnych bitów niektórych rejestrów z obszaru **wewnętrznej pamięci danych** procesora.

Należy wiedzieć, że nie wszystkie z 256 rejestrów RAM procesora (128 użytkownika + 128 SFR) „dają się” w ten sposób zapisywać lub odczytywać.

Spośród rejestrów specjalnych SFR do takich, które można modyfikować bit po bicie należą:

– rejestr B

- akumulator Acc
- słowo stanu PSW
- rejestr sterujący licznika T2: T2CON
- rejestr priorytetu przerwań: IP
- rejestr maski przerwań: IE
- rejestr sterujący portem szeregowym: SCON
- rejestr sterujący licznikami T0, T1 oraz INT0 i INT1: TCON
- oczywiście rejestry portów I/O procesora: P0, P1, P2, P3

Każdy bit rejestru adresowalnego bitowo posiada swój adres, tak jak ponumerowane są rejestry w wewn. RAM procesora. W jaki sposób można więc rozróżnić adresowanie rejestrów od adresowania bitów. To proste, w liście instrukcji procesora 8051 istnieją po prostu instrukcje specjalne operujące na bitach, przedstawiłem je kilka numerów temu, kiedy to omawialiśmy instrukcje procesora – zapraszam do powtórki.

Oprócz niektórych rejestrów SFR, także część rejestrów z obszaru wewnętrznej pamięci użytkownika, tj. o adresach 00...7Fh także daje się adresować. Są to rejestry o adresach: 20h...2Fh. Rejestrów jest więc 16, czyli bitów do adresowania jest  $16 \times 8 = 128$ .

Rozkład adresów poszczególnych bitów tych rejestrów jest bardzo prosty, oto on:

| adres bajtu | adresy bitów (HEX) |    |    |    |    |    |    |    |
|-------------|--------------------|----|----|----|----|----|----|----|
| <b>2Fh</b>  | 7F                 | 7E | 7D | 7C | 7B | 7A | 79 | 78 |
| <b>2Eh</b>  | 77                 | 76 | 75 | 74 | 73 | 72 | 71 | 70 |
| <b>2Dh</b>  | 6F                 | 6E | 6D | 6C | 6B | 6A | 69 | 68 |
| .....       |                    |    |    |    |    |    |    |    |
| .....       |                    |    |    |    |    |    |    |    |
| .....       |                    |    |    |    |    |    |    |    |
| .....       |                    |    |    |    |    |    |    |    |
| .....       |                    |    |    |    |    |    |    |    |
| <b>21h</b>  | 0F                 | 0E | 0D | 0C | 0B | 0A | 09 | 08 |
| <b>20h</b>  | 07                 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |

.....  
..... itd.  
.....

Jeżeli chodzi o adresy bitów rejestrów SFR procesora 8051, to przedstawiam je poniżej.

| adres rejestru | adresy bitów (HEX) |    |    |    |    |    |    |    | nazwa rejestru |
|----------------|--------------------|----|----|----|----|----|----|----|----------------|
| <b>F0h</b>     | F7                 | F6 | F5 | F4 | F3 | F2 | F1 | F0 | <b>B</b>       |
| <b>E0h</b>     | E7                 | E6 | E5 | E4 | E3 | E2 | E1 | E0 | <b>Acc</b>     |
| <b>D0h</b>     | D7                 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | <b>PSW</b>     |
| <b>C8h</b>     | CF                 | CE | CD | CC | CB | CA | C9 | C8 | <b>T2CON</b>   |
| <b>B8h</b>     | –                  | –  | BD | BC | BB | BA | B9 | B8 | <b>IP</b>      |
| <b>B0h</b>     | B7                 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | <b>P3</b>      |
| <b>A8h</b>     | AF                 | –  | AD | AC | AB | AA | A9 | A8 | <b>IE</b>      |
| <b>A0h</b>     | A7                 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | <b>P2</b>      |
| <b>98h</b>     | 9F                 | 9E | 9D | 9C | 9B | 9A | 99 | 98 | <b>SCON</b>    |
| <b>90h</b>     | 97                 | 96 | 95 | 94 | 93 | 92 | 91 | 90 | <b>P1</b>      |
| <b>88h</b>     | 8F                 | 8E | 8D | 8C | 8B | 8A | 89 | 88 | <b>TCON</b>    |
| <b>80h</b>     | 87                 | 86 | 85 | 84 | 83 | 82 | 81 | 80 | <b>P0</b>      |

Jak widać z obu tabel, adres rejestrów użytkownika i SFR nie pokrywają się, co umożliwia jednoznaczne rozróżnienie ich podczas adresowania bitowego. I tak np. wydanie polecenia:

```
SETB 0AFh
```

spowoduje ustawienie bitu 7 (najstarszego) w rejestrze IE, czyli bitu o nazwie EA – odblokowującego przerwania.

Operacje tę można przeprowadzić także za pomocą modyfikacji całego rejestru, ale nie będzie to już adresowanie bitowe:

```
ANL IE, #7Fh
```

Inny przykład, wydanie polecenia:

```
MOV 0Eh, C
```

spowoduje w efekcie skopiowanie bitu C, czyli bitu 7 w słowie PSW (adres bitu: D7h) do bitu 6 w rejestrze o adresie 21h w wewnętrznej RAM – obszar rejestrów użytkownika. Kwestie adresowania bitowego poruszę w następnym odcinku klasy mikroprocesorowej.

Na razie radzę przypomnieć sobie zasady dotyczące adresowania i operacji na bitach z odcinków kursu poświęconych liście rozkazów procesora 8051.

Sławomir Surowiński

W dzisiejszym odcinku traktującym o mikrokontrolerach 8051/52 zapoznamy się z możliwościami i sposobami obsługi i programowania tych wersji kostek które posiadają wewnętrzną pamięć programu typu EPROM. Ze względu na to, że w ostatnich latach, cena układów w takich wersjach bardzo spadła, nawet kilkanaście razy, oraz pojawiło się wiele mutacji procesorów C51/52 z pamięcią reprogramowalną typu „Flash” EEPROM, stały się one dostępne dla większości hobbystów – amatorów techniki mikroprocesorowej. Jeżeli nawet nie zamierzasz wkrótce korzystać z dobrodziejstw procesora w wersji z wewnętrzną, reprogramowalną pamięcią programu, to przedstawiona w artykule garść informacji z pewnością, przyczyni się do większego oswojenia się z popularną ‘51-ką.



Pamiętam jeszcze czasy, a było to prawie dekadę temu, kiedy to cena procesora 87C51 była tak duża, że aby go zdobyć, musiałem sporo odkładać ze studenckiego stypendium. Kiedy wreszcie udało mi się kupić wymarzoną kostkę, środków ostrożności nie było nigdy za wiele. A to nosiło się układ w folii aluminiowej, a przed wyjęciem wyrównywało się swój – ludzki potencjał dotykając kaloryfera lub rury wodociągowej, a to sprawdziło się zmontowaną płytkę drukowaną przed włożeniem drogiego układu kilka razy. Wszystko po to aby przypadkiem

nie uszkodzić delikatnej struktury mikroprocesora. Dla przykładu podam, że wówczas przy cenie procesora 87C51 równej około 500 tys. złotych (50,- nowych złotych), wersja bez pamięci EPROM lub wersje z pamięcią stałą ROM, kosztowały około 20...30 tys. (2...3 nowe złote), czyli około 20 razy mniej. Nie wspominam tutaj o procesorach takich jak np. 87C52 z wewnętrzną pamięcią EEPROM, których zdobycie było nie lada trudnością, a jeżeli już były to trzeba było za nie zapłacić ponad milion starych złotych, czyli prawie 1/3 ówczesnej przeciętnej pensji!

Tabela 1

| Symbol handlowy | Opis                                                                                                                                                                                                                                                                                 |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 80C51           | wersja z wewnętrzną pamięcią programu typu ROM, której zawartość jest niezmienna z naszego punktu widzenia, toteż układ możemy wykorzystać do pracy tylko z dołączoną zewnętrzną pamięcią np. EPROM do której zapiszemy nasz program (wtedy pamięć ROM jest wyłączona – nieaktywna). |
| 80C31           | wersja procesora bez wewnętrznej pamięci programu. Mikrokontroler w tej wersji może pracować tylko z dołączoną zewnętrzną pamięcią jak dla 80C51.                                                                                                                                    |
| 87C51           | wersja z wbudowaną pamięcią EPROM – 4kB. Obudowa mikroprocesora posiada okienko kwarcowe, dzięki któremu możliwe jest kasowanie zawartości tej pamięci, co umożliwia wielokrotne programowanie całego układu.                                                                        |
| 89C51           | wersja procesora z kasowaną elektrycznie pamięcią EEPROM – 4kB. Ponieważ w tej wersji cała pamięć programu EEPROM może być kasowana bardzo szybko – za pomocą tylko 1 impulsu, procesory w tej wersji nazywa się typu „Flash”.                                                       |
| 80C52           | jest to procesor identyczny z 8051 tyle, że posiada dodatkowy 3-ci programowalny licznik/timer (nazywany jako „T2”). Reszta jak dla 80C51 – patrz wyżej.                                                                                                                             |
| 80C32           | jak dla 80C31 z uwzględnieniem „T2”.                                                                                                                                                                                                                                                 |
| 87C52           | jak dla 87C51 z uwzględnieniem „T2”, wewn. pamięć programu ma 8kB.                                                                                                                                                                                                                   |
| 89C52           | jak dla 89C51 z uwzględnieniem „T2”, wewn. pamięć programu ma 8kB.                                                                                                                                                                                                                   |

Teraz, kiedy na rynku aż roi się od mikroprocesorów w różnych wersjach, ceny układów znacznie spadły. Obecnie za kwotę 15 zł można kupić popularną ‘51-kę z pamięcią EEPROM „Flash”, a kostki tzw. „ROM less”, czyli bez wewnętrznej pamięci programu lub z fabrycznym ROMem, spotka się za kilka złotych. I właśnie ze względu na to, że różnica między cenami układów jest taka mała, warto zainteresować się takim właśnie wersjami tych jakże popularnych kostek.

W jednym z pierwszych odcinków szkoły mikroprocesorowej, opisywałem kilka najpopularniejszych obecnie wersji procesorów oraz ich przybliżone ceny. Ponieważ było to ponad rok temu, obecne ceny tych układów są jeszcze niższe. Dla przypomnienia zamieszczam informacje dotyczące najpopularniejszych obecnie mikrokontrolerów serii MCS-51 (patrz tabela 1).



## Też to potrafisz

Zanim przejdę do omówienia sposobów programowania i weryfikacji pamięci wewnętrznej programu mikrokontrolerów '51, powinienem Ci uzmysłowić drogi Czytelniku, że do wykonania tej operacji będzie potrzebny, oprócz dobrej woli, także „programator”, w dodatku nie byle jaki, bo potrafiący programować procesory rodziny MCS-51.

Na rodzimym rynku można znaleźć sporo urządzeń tego typu, kosztujących od kilkuset złotych do kilkudziesięciu! Nie oznacza to że aby zaprogramować procesor trzeba wybrać urządzenie najdroższe, chodzi o to aby znaleźć te tańsze, potrafiące jednak bez żadnych przystawek, fachowo nazywanych adapterami, programować chociaż podstawowe procesory z rodziny MCS-51, a więc: 87C51, 87C52, 98C51, 98C52. Na szczęście większość amatorskich programatorów oferowanych przez drobnych rodzimych wytwórców, spełnia te wymagania, a ich cena nie zwala z nóg przeciętnego „zjadacza chleba” który interesuje się techniką mikroprocesorową i chce dokształcić się w tej jakże interesującej dziedzinie wiedzy.

Niestety chęć użyczenia kostek z wewnętrzną pamięcią programu, oprócz posiadania programatora, wymaga także posiadania komputera PC. Tak więc „ręczniakom” należą się w tym miejscu przeprosiny, lecz musicie zdawać sobie sprawę drodzy koledzy, że aby osiąść stosowną wiedzę, każdy z nas jest zdolny do wielu wyrzeczeń. Tak też było w moim przypadku, aczkolwiek jeszcze kilka ładnych lat temu, kiedy królowały komputery PC typu XT oraz AT, kupno jednego z nich było nie lada wysiłkiem dla całej mojej rodziny. W chwili obecnej komputer PC wystarczający do obsługi programatora procesorów nawet w najmniejszej konfiguracji powinien mieć procesor co najmniej 80286, 1MB pamięci RAM, oraz jakikolwiek twardy dysk z wolnymi ok. 2,5MB przestrzeni. Takiej konfiguracji praktycznie nie spotka my już na rynku, z pomocą przyjdzie musza więc giełdy, gdzie proponowany zestaw w nieco lepszej (z procesorem 80386) konfiguracji można nabyć na 200..300 zł. Do tego należy dokupić jeszcze używany monitor mono za około 50 zł i można zabrać się do programowania. Ważne jest aby przy takiej konfiguracji komputera program obsługi wybranego przez Ciebie programatora potrafił pracować w trybie tekstowym, bowiem zainstalowanie systemu Windows tylko dla celów programowania '51-ek na tym etapie wiedzy nie ma za bardzo sensu.

Na szczęście większość oprogramowania na dostępne na naszym rynku programatory pracuje w środowisku tekstowym MS-DOS, i posiada bardzo ograniczone wymagania sprzętowe co do komputera, toteż z instalacją nabytego urządzenia nie powinno być większych problemów.

Na pocieszenie pragnę poinformować, że w ofercie handlowej AVT znajduje się idealny do naszych potrzeb programator przeznaczony specjalnie dla rodziny procesorów MCS-51. Urządzenie posiada kod handlowy AVT-320 i sprzedawane jest w postaci zestawu do samodzielnego montażu (wersja /B) lub jako zmontowane (wersja /C). Bardziej zaawansowani i wytrwali elektronicy mogą też nabyć samą płytkę drukowaną wraz z kilkoma układami scalonymi opracowanymi specjalnie dla potrzeb tego urządzenia (zestaw /A). Dołączona do zestawu dyskietka zawiera prosty ale funkcjonalny program obsługi programatora. Urządzenie współpracuje z komputerem typu PC (począwszy od pocziwego XT na szybkich Pentiumach skończywszy) poprzez port szeregowy RS232C.

Wszystkich zainteresowanych zachęcam do lektury artykułu na ten temat, który ukazał się w naszym bratnim piśmie – „Elektronice Praktycznej” w numerach 9,10 i 11/97, a jest autorstwa niżej podpisanego.

## Obsługa pamięci programu

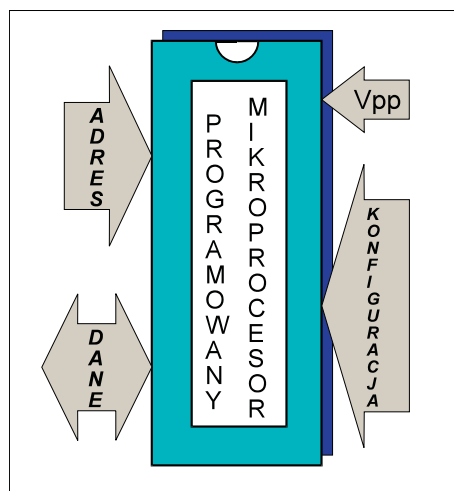
W tym miejscu powinienem wyjaśnić że w tytule tej części artykułu chodziło mi o wspomnianą wcześniej sprawę programowania pamięci wewnętrznej procesora. Ale na samym programowaniu się nie kończy, bowiem po tej operacji trzeba zapisać pamięć sprawdzić – czyli fachowo mówiąc „zweryfikować”. Istnieje jeszcze kilka dodatkowych operacji, które jednak omówię w dalszej części artykułu.

Zanim przejdę do omówienia sposobów programowania pamięci programu, poinformować niektórych z Was, drodzy Czytelnicy, że w zasadzie podane niżej informacje nie są niezbędne do posługiwania się kontrolerami z wbudowaną pamięcią programu i korzystania z programatorów tych kostek. Jednak aby w pełni zrozumieć zasadę działania i dodatkowe funkcje procesora, warto znać te wiadomości, a praktyka przyniesie z pewnością mniej niemiłych niespodzianek.

Oto kilka informacji, które pozwolą zrozumieć Ci wstępnie w jaki sposób można zaprogramować wewnętrzną pamięć programu procesora.

1. Wiesz, już w jaki sposób procesor wykonuje swój program, i że do tego z jakiej pamięci (zewnętrznej czy wewnętrznej) odczytywany jest program, służy końcówka /EA (pin 31). W przypadku kiedy wyprowadzeni to jest zwarte do masy procesor pobiera rozkazy z zewnętrznej pamięci programu o adresach 0000h...FFFFh, czyli maksymalnie z 64kB (65536 bajtów). W przypadku kiedy zewrzymy to wyprowadzenie do plusa zasilania (+5V) uaktywniona zostanie wewnętrzna pamięć programu (w kostkach 87C51, 89C51 lub podobnych) i kolejne rozkazy będą pobierane właśnie z niej.
2. Fizycznie wewnętrzną pamięć programu można wyobrazić sobie jako wbudowany w procesor układ reprogramowalnej pamięci EPROM (87C51) lub EEPROM (np. 89C51), który za pośrednictwem zewnętrznych końcówek procesora może być zaprogramowany przez urządzenie zewnętrzne – programator.
3. Ponieważ procesor posiada te same i niezmiennie wyprowadzenia (40 dla omawianych kostek 8051/C51, 87C51, 89C51/C52) programowanie i weryfikacja wewnętrznej pamięci programu odbywa się z wykorzystaniem tych samych wyprowadzeń, tylko że w tzw. „trybie programowania”. W trybie tym procesor znajduje się poza układem macierzystym (tym w którym ma pracować) a umieszczony jest w pro-

Rys. 1. Ogólny sposób na programowanie procesora





gramatorze, który w odpowiedni sposób sterując pewnymi wyprowadzeniami kostki wprowadza ją w ten właśnie tryb.

- Skoro powiedziałem o tym że wewnętrzną pamięć programu można wyobrazić sobie jako wbudowany chip pamięci EPROM/EEPROM, to oznacza to że do „dobrania się” do niej muszą służyć:

- linie adresowe, których liczba zależy od wielkości pamięci programu
- linie danych : w procesorach takich jak MCS-51 będzie ich oczywiście 8
- dotatkowe linie konfiguracyjne – sterujące zapisem i odczytem tej pamięci. Obrazowo pokazano to na **rysunku 1**

- Wśród tych ostatnich – linii sterujących znajduje się także tzw. linia Vpp – czyli linia napięcia programującego. Z reguły napięcie to jest wyższe od napięcia zasilania procesora i wynosi:

- 12,75 V dla większości układów z pamięcią EPROM, np. 87C51/2
- 12V dla większości układów z pamięciami EEPROM (Flash EEPROM), np. 89C51/2. Istnieją także wersje programowane napięciem 5V (np. procesory Atmela o oznaczeniach 89C51-XX-5, gdzie XX oznacza maksymalną częstotliwość pracy układu w MHz).

W zamierzonych czasach istniały także wersje procesorów 8751 które programowano napięciem 21V, podobnie jak pamięci EPROM wykonywane kiedyś w technologii MOS (obecnie CMOS), ale to przeszłość i takich wersji układów na rynku się nie spotyka.

- Dzięki stosownemuysterowaniu wspomnianych końcówek procesora w trybie programowania, a następnie poprzez podanie napięcia programującego Vpp stosowna zaadresowana przez programator komórka w wewnętrznej pamięci programu zostaje zaprogramowana. Po obniżeniu napięcia Vpp do wartości napięcia zasilającego Vcc (+5V) programator może zweryfikować zapisany bajt. Tak z grubsza odbywa się każdy cykl zapisu i sprawdzenia poprawności zaprogramowanej komórki.
- Do prawidłowego programowania procesora potrzebny jest także dołączony do końcówek XTAL1 i XTAL2 rezonator kwarcowy, tak aby procesor mógł „oddychać” podczas programowania. Już wiesz przecież że bez sygnału zegarowego

procesor jest „martwy” jak człowiek bez krwi. Wartość częstotliwości rezonansowej kwarcu nie jest w tym przypadku istotna, ważne jest aby zawierała się ona w następujących granicach:

- 4...6 MHz dla układów z pamięcią EPROM (87C51/C52)
- 4...20 MHz dla układów z pamięcią EEPROM/ Flash (89C51/C52)

- Ktoś może w tym momencie zapytać: „...No dobrze, procesor pracując w trybie z zewnętrzną pamięcią programu może czytać rozkazy spod adresów 0000h...FFFFh, a ile jest tej pamięci wewnętrznej programu?...”, a no tyle ile podałem w tabeli 1.

I tak dla poszczególnych kostek :

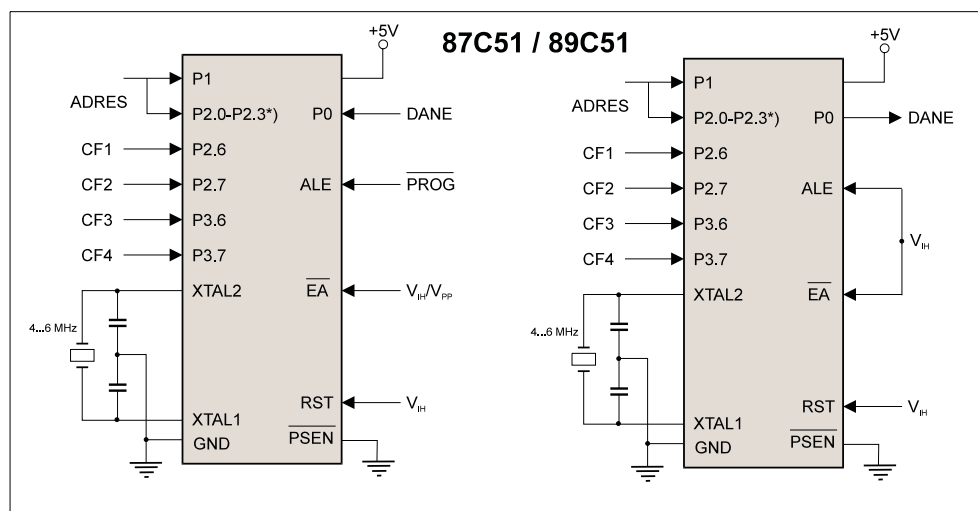
- 87C51, 89C51 jest to 4kB (4096 bajtów), adresy 0000h...0FFFh
- 87C52, 89C52 jest to 8kB (8192 bajty), adresy 0000h...1FFFh

- Inne pytanie – „...a co się stanie, kiedy to np. pracując w trybie z wewnętrzną pamięcią programu (końcówka /E-A zwarta do Vcc) program dojdzie do końca obszaru wewnętrznej pamięci programu (np. do adresu 0FFFh dla kostki 87C51), co będzie wtedy, skąd będą pobierane dalsze rozkazy?...”. Jeżeli tak się stanie i procesor dojdzie do końca tej pamięci to dalsze rozkazy będą pobierane z zewnętrznej pamięci programu dołączonej do procesora w tradycyjny (jak w naszym komputerku edukacyjnym AVT-2250) sposób.

- Na **rysunku 2** pokazano sposób dołączenia sygnałów : adresowych, danych i sterujących w tym napięcia Vpp podczas programowania wewnętrznej pamięci programu kostek. W przypadku programowania procesorów w wersji z pamięci programu EEPROM „Flash” typu 89C51 /C52 na rysunku będą pewne różnice, a mianowicie: rezonator kwarcowy może być z zakresu 4...20 MHz
- W przypadku układów 87C52 i 89C52 adres komórki do zaprogramowania podawany jest na linie portów : P1 – LSB adresu oraz P2.0 – P2.4 : MSB adresu
- Przy programowaniu układów 87C51/52 na wejście sterujące /PROG (końcówka ALE) podczas programowania danej komórki pamięci podaje się 25 impulsów ujemnych (od Vcc do masy) o czasie trwania min. 100µs i przerwie ok. 10µs.

- Przy programowaniu kostek 89C51 i 89C52 na wejście /PROG wystarczy podać 1 impuls ujemny o czasie trwania ok. 100µs, to wystarczy aby zaprogramować bajt w wewnętrznej pamięci programu tej kostki
- 10. Warto wiedzieć, że zapisaną, wewnętrzną pamięć programu można zabezpieczyć przed odczytem przez osoby niepowołane. Służą temu tzw. „Security Bits”, czyli bity zabezpieczające, których odpowiednie „przepalenie” uniemożliwia odczytanie zawartości pamięci (nie bójcie się, nie robi się tego „zapalniczką”, chodzi mi tu o ich zaprogramowanie).

Rys. 2. Sposób dołączenia sygnałów sterujących podczas programowania i weryfikacji układów 87C51/52.



Uwaga \*) : W przypadku kostki 87C52, wykorzystana jest dodatkowa linia adresowa – końcówka P2.4, ze względu na większą – 8kB pamięć programu.

## Też to potrafisz

mowanie). Zabezpieczenia zawartości programu może być często użyteczne, kiedy np. mamy zamiar oferować osobom drugim swój zaprogramowany mikroprocesor pracujący w mniej lub bardziej wymyślnym urządzeniu, sprzedając go i nie chcąc jednocześnie aby ktoś skopiował nasz pomysł i powielił w setkach tysięcy egzemplarzy.

I tu kryje się istotna zaleta procesorów z wewnętrzną pamięcią programu. Otóż zauważmy, że w przypadku umieszczenia programu w zewnętrznej pamięci EPROM, praktycznie każdy ma do niej dostęp, i może korzystając z programatora pamięci EPROM odczytać jej zawartość, w celu późniejszego skopiowania. Takie postępowanie jest oczywiście niezgodne z prawem, ale kto jest w stanie dochodzić swoich praw, szczególnie, że program może zostać sprytnie zmodyfikowany przez programistę-pirata w sposób uniemożliwiający późniejsze udowodnienie mu jego winy. Najistotniejsze jest to że w takich sytuacjach nasz, często opracowywany miesiącami pomysł zostanie błyskawicznie skradziony i powielony!

11. I tu z pomocą przychodzi wewnętrzna pamięć programu i bity ją zabezpieczające. Otóż raz zapisany i zabezpieczony program w procesorze jest nie do odczytania! Nie ma sposobu aby program taki odczytać jak ze zwykłej pamięci EPROM. Jeżeli ty właśnie jesteś autorem tego programu, to i tak wszystko w porządku, bo przechowujesz gdzieś, zapewne w komputerze, kopię programu oraz listing źródłowy. A potencjalny pirat? – ten musi obejść się smakiem, bo i tak nic nie wskóra, a program będzie zabezpieczony przed jego ingerencją.
12. Pytanie: „...No tak, ale skoro zabezpieczymy już ten program, to jak go potem usunąć? Zwyczajnie. W układach

Tabela 2

| Tryb dla 87C51/C52       | RST | PSEN | ALE PROG | EA/Vpp | CF1 P2.6 | CF2 P2.7 | CF3 P3.6 | CF4 P3.7 |
|--------------------------|-----|------|----------|--------|----------|----------|----------|----------|
| Zapis danych             | 1   | 0    | 0*       | Vpp    | 0        | 1        | 1        | 1        |
| Odczyt danych            | 1   | 0    | 1        | 1      | 0        | 0        | 1        | 1        |
| Bity zabezpieczające B1  | 1   | 0    | 0*       | Vpp    | 1        | 1        | 1        | 1        |
| Bity zabezpieczające B2  | 1   | 0    | 0*       | Vpp    | 1        | 1        | 0        | 0        |
| Prog. tabeli szyfrującej | 1   | 0    | 0*       | Vpp    | 0        | 1        | 0        | 1        |
| Odczyt sygnatury układu  | 1   | 0    | 1        | 1      | 0        | 0        | 0        | 0        |

Uwagi:

- a) 0\* oznacza że należy podać 25 impulsów ujemnych o czasie trwania ok. 100µs i czasie przerwy min. 10µs
- b) Vpp = 12,75 V ± 0,25V
- c) Vcc = 5V ± 10% (dla programowania i weryfikacji)

Tabela 3

| Tryb dla 89C51/C52         | RST | PSEN | ALE PROG | EA/Vpp | CF1 P2.6 | CF2 P2.7 | CF3 P3.6 | CF4 P3.7 |
|----------------------------|-----|------|----------|--------|----------|----------|----------|----------|
| Zapis danych               | 1   | 0    | 0*       | Vpp    | 0        | 1        | 1        | 1        |
| Odczyt danych              | 1   | 0    | 1        | 1      | 0        | 0        | 1        | 1        |
| Bity zabezpieczające B1    | 1   | 0    | 0*       | Vpp    | 1        | 1        | 1        | 1        |
| Bity zabezpieczające B2    | 1   | 0    | 0*       | Vpp    | 1        | 1        | 0        | 0        |
| Bity zabezpieczające B3    | 1   | 0    | 0*       | Vpp    | 0        | 1        | 1        | 1        |
| Kasowanie pamięci programu | 1   | 0    | 0*       | Vpp    | 0        | 1        | 1        | 1        |
| Odczyt sygnatury układu    | 1   | 0    | 1        | 1      | 0        | 0        | 0        | 0        |

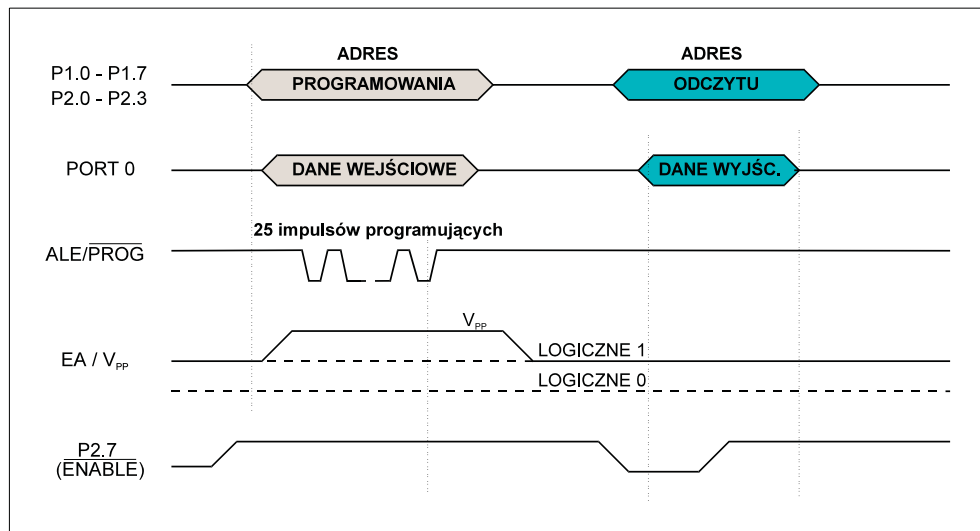
Uwaga:

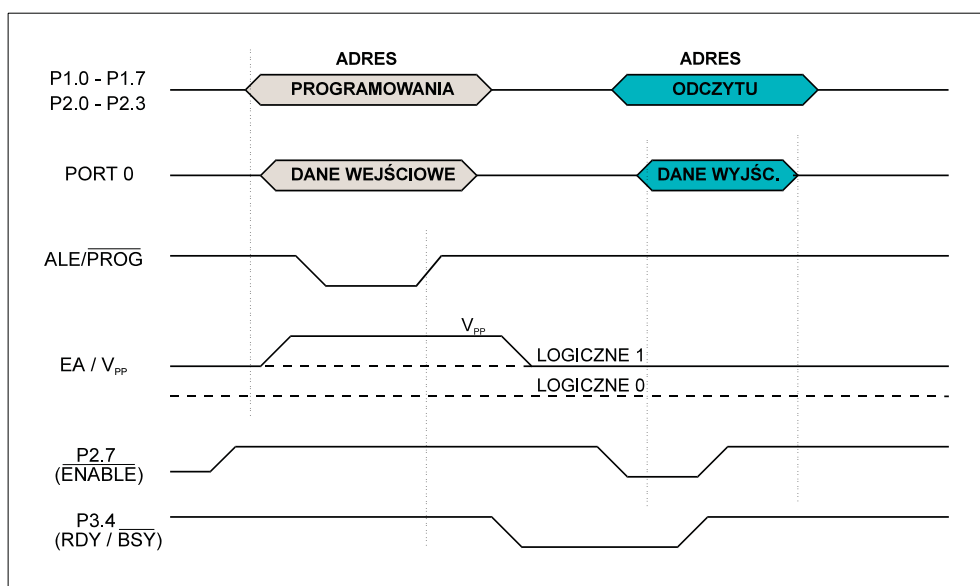
- a) 0\* oznacza że należy podać 1 impuls ujemny o czasie trwania ok. 100µs
- b) Vpp = 12 V ± 0,25V lub 5V ± 0,25V dla wersji 89C51/C52 – XX – 5 (patrz tekst)
- c) procedura kasowania pamięci wymaga podania impulsu na wejście /PROG o czasie trwania 10 ms.

z pamięcią EEPROM robi się to tak jak w przypadku pamięci EPROM, korzystając z kwarcowego, przezroczystego okienka. Procesor umieszcza się po prostu w kasowniku ultrafioletowym i po około 15 minutach jest po wszystkim, kostka jest „czysta” i gotowa do ponownego zaprogramowania i użycia.

Inaczej jest w przypadku nowocześniejszych układów z pamięcią EEPROM „Flash”. Tutaj nie jest potrzebne promieniowanie ultrafioletowe. Kasowanie wewnętrznej pamięci programu odbywa się w programatorze. Wysterowując końcówki sterujące, podając napięcie Vpp (patrz tabela 3) oraz impuls ujemny o czasie trwania ok. 10 ms, powodujemy skasowanie całej zawartości pamięci programu. Po takiej operacji układ jest gotowy do ponownego zaprogramowania i użycia.

Rys. 3. Zależności czasowe podczas programowania procesorów a pamięcią EPROM (87C51, 87C52).





Rys. 4. Zależności czasowe podczas programowania procesorów a pamięcią EEPROM/Flash (89C51, 89C52).

W zależności od ustawienia sygnałów sterujących oznaczonych na rysunku 2 jako CF1...CF4 oraz dodatkowych ALE i EA, RST i PSEN można uzyskać kilka funkcji programowania lub weryfikacji wewnętrznej pamięci programu. Wszystkie dozwolone kombinacje przedstawiają:

- dla kostek 87C51/ C52 – **tabela 2**
- dla kostek 89C51/ C52 – **tabela 3**

W dalszej części artykułu dokładnie objaśnię znaczenie poszczególnych pozycji tabel 2 i 3 podczas operacji programowania wewnętrznej pamięci mikrokontrolerów.

13. Wątpliwość: „... No tak ale przecież istnieje w liście rozkazów procesora instrukcja `MOVC A,@A+DPTR`, dzięki której możliwe jest odczytanie każdego bajtu z wewnętrznej czy zewnętrznej pamięci programu, co wtedy...? Odpowiadam: i na to jest rada. Otóż producenci procesorów umieścili dodatkowy bit zabezpieczający, którego „przepalenie” (zaprogramowanie) powoduje zablokowanie tej instrukcji, w wypadku kiedy ktoś próbuje wykonać ją z obszaru zewnętrznej pamięci programu – zastanów się dlaczego jest to dobry sposób na zabezpieczenie?

14. I na koniec jeszcze jedna informacja o zabezpieczeniach. Otóż niektórzy producenci procesorów rodziny MCS-51, prawie wszyscy produkujący układy w wersji z pamięcią EPROM stosują dodatkowe zabezpieczenia w postaci tzw. tablicy szyfrującej (ang. Encryption Table). Fizycznie jest to wydzielona część pamięci EPROM, o rozmiarze przeważnie równym 1, 2 lub 4 krotności 16 bajtów. Zaprogramowanie tabeli szyfrującej sekwencją 16, 32 lub 64 bajtów (zależnie od wersji układu) po uprzednim zaprogramowaniu wewnętrznej pamięci programu, powoduje, że w przypadku nie zabezpieczenia procesora bitami zabezpieczającymi, odczytywany przez potencjalnego hackera każdy bajt programu będzie wynikiem operacji EXNOR („Exclusive NOR”) faktycznego bajtu programu z kolejnym (modulo wielkość tabeli szyfrującej) bajtem tabeli szyfrującej. Dzięki temu bez znajomości zawartości tabeli enkrypcji (która po zaprogramowaniu nie jest dostępna) nie jest praktycznie możliwe rozkodowanie programu przez osobę nie mającą dostępu do zawartości tabeli szyfrującej. Oczywiście autor programu posiada takową kopię i wie co trzeba zrobić z odczytanym programem aby doprowadzić go do stanu „używalności”. W ostatnich czasach, ze względu na niepotrzebną często komplikację, większość producentów procesorów rodziny MCS-51 odeszła od koncepcji stosowania tabeli szyfrującej i stosuje dodatkowe bity zabezpieczające, które pokrótce opiszę w dalszej części artykułu.

Na **rysunku 3 i 4** przedstawiłem zależności czasowe pomiędzy sygnałami sterującymi podczas programowania procesorów z pamięciami EPROM (87C51/C52) i EEPROM „Flash” (89C51/C52).

## Charakterystyka pamięci EPROM/EEPROM procesorów

Układy 87C51/C52 wyposażone są w pamięć EPROM wykonaną w technologii CMOS, programowana tzw. algorytmem szybkim „Quick-Pulse Programming”. Algorytm ten polega na podaniu napięcia  $V_{pp}$  o wysokości 12,75V (na wejście EA/ $V_{pp}$ ) a następnie podanie na wejście ALE procesora serii 25 impulsów ujemnych o czasie trwania 100µs (stan L) i przerwie min. 10µs (stan H).

W wypadku procesorów 89C51/ C52 napięcie  $V_{pp}$  może mieć jedną z dwóch wartości: 12V i 5V w zależności od wersji procesora (patrz tekst wyżej).

Mikrokontrolery posiadają tzw. sygnatury, dzięki którym możliwa jest autoidentyfikacja układu przez obsługujący programator. Fizycznie są to pojedyncze komórki ROM wbudowane w procesor z zapisanymi bajtami mówiącymi o producencie układu, jego typie oraz wersji wykonania. Są to:

a) dla układów z pamięcią EPROM

- (adres: 030h) = 15h, oznacza producenta (w tym wypadku jest to Philips)

Tabela 4

| Tryb | LB1 | LB2 | LB3 | Rodzaj zabezpieczenia                                                                                                                                                                                |
|------|-----|-----|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    | U   | U   | U   | Program nie zabezpieczony                                                                                                                                                                            |
| 2    | P   | U   | U   | Ignorowane są instrukcje <code>MOVC</code> , wykonywane z pamięci zewnętrznej programu, wejście EA jest zatrzęsnięte podczas „resetu” procesora, zablokowane jest dalsze programowanie pamięci Flash |
| 3    | P   | P   | U   | Tak jak w trybie 2 z zablokowaną możliwością weryfikacji programu                                                                                                                                    |
| 4    | P   | P   | P   | Tak jak w trybie 3 z zablokowaną możliwością pobierania rozkazów z zewnętrznej pamięci programu.                                                                                                     |

LB1 - bit nr 1, LB2 - bit nr 2, LB3 - bit nr 3

P - bit zaprogramowany  
U - bit niezaprogramowany

## Też to potrafisz

- (adres: 031h) = 92h, oznacza układ 87C51, podobnie dla 87C52 jest to 97h
- b) dla układów z pamięcią EEPROM / Flash (np. producenta – firmy Atmel)
- (adres: 030h) = 1Eh, oznacza producenta w tym przypadku Atmel)
  - (adres: 031h) = 51h, oznacza układ 89C51
  - (adres: 032h) = FFh oznacza napięcie  $V_{pp}=12V$ , 05h oznacza  $V_{pp}=5V$ .

Ze względu na różnorodność typów układów jak i braku jednolitego standardu wśród producentów, podane wartości mogą się zmieniać. Należy więc je traktować jako informacyjne. Na zakończenie wspomnę że dostęp do sygnatury odbywa się poprzez zaadresowanie (adres podany w nawiasie) sygnatury tak jak to się odbywa w przypadku weryfikacji pamięci programu, z tą różnicą, że odmienny jest układ sygnałów sterujących – patrz tabele 2 i 3.

Do prawidłowego zaprogramowania kości potrzebny jest dołączony zewnętrzny oscylator kwarcowy (patrz rys.2) Powodem zastosowania tego elementu jest fakt, że podczas programowania, pracuje licznik wewnętrznego adresu procesora oraz odbywa się transfer danych z rejestrów portów procesora do pamięci programu.

Podczas programowania, adres danej komórki EPROM / EEPROM podawany jest przez programator na port P1 procesora (młodsza część adresu – LSB) oraz na część pinów portu P2. Przy układach 'C51 wyposażonych w 4 kB pamięci programu są to linie P2.0... P2.3, a w kostkach 'C52 dodatkowo sterowana jest linia P2.4. W ten sposób dzięki 12 liniom adresowym (A0...A11) dla 'C51 oraz 13 liniom adresowym (A0...A12) dla kostek 'C52 możliwe jest zaadresowanie całej wewnętrznej pamięci programu.

Jeżeli chodzi o dane to programator podaje je na port P0 procesora. Następnie ustawione zostają (zgodnie z tabelą 2 dla układów z EPROM oraz tabelą 3 dla układów z EEPROM Flash) sygnały RST i PSEN oraz wybrana zostaje konfiguracja pinów CF1...CF4 określających zgodnie z tymi tabelami operacje na procesorze jaka ma być właśnie wykonana. Przy programowaniu sekwencja poziomów logicznych sygnałów CF1...CF4 będzie równa: 0-1-1-1.

Następnie (patrz zależności czasowe na rys.3 i 4) programator podaje napięcie  $V_{pp}$  na końcówkę EA procesora, po czym po krótkiej chwili, kiedy napięcie to narosnie do odpowiedniej wartości programator

- dla układów z EPROM : generuje szereg impulsów programujących (25) o parametrach jak podałem wcześniej w artykule
- dla układów z EEPROM jest to 1 impuls o określonym czasie trwania (zazwyczaj jest to 100µs)

Następnie programator obniża do pierwotnej wartości  $V_{cc}$  wartość napięcia programującego  $V_{pp}$ , po czym wystawia poziom niski na linię CF2, co powoduje że zapisana przed chwilą dana jest wystawiana, tym razem przez procesor na linie portu P0 celem weryfikacji (odczytu) przez urządzenie programujące.

W układach 89C51/52 programator może monitorować stan programowania komórki za pośrednictwem dodatkowej linii P3.4 procesora. Otóż po zapisaniu danej w pamięci wewnętrznej programu, procesor sygnalizuje to pojawieniem się stanu niskiego na tej linii, co może odczytać programator i w ten sposób skrócić niezbędny czas impulsu programującego, podawanego na wejście /PROG procesora.

Programowanie tablicy szyfrującej („Encryption Table”) w procesorach z pamięcią EPROM odbywa się podobnie, jak w przypadku programowania pamięci programu, lecz inna jest kombinacja sygnałów sterujących CF1...CF4 (patrz tabela 2).

Podobnie wygląda programowanie bitów zabezpieczających, z tą różnicą, że do przepalenia danego bitu wystarczy kombinacja sygnałów CF1...CF4 oraz jak poprzednio cykl programujący z  $V_{pp}$  (jak poprzednio). Linie adresowe oraz danych nie mają w tym momencie znaczenia.

W tabeli 4 przedstawione są efekty przepalania kolejnych bitów zabezpieczających. W przypadku układów 87C51/C52 mamy do czynienia tylko z dwoma bitami LB1 i LB2. Przepalenie LB1 zabezpiecza układ przed przyszłym programowaniem, czy raczej „doprogramowaniem” tej części pamięci, która nie została wcześniej podczas programowania zapisana (nie musimy przecież programować całej pamięci programu, a tylko tyle ile ma nasz program. Przepalenie bitu LB2 powoduje zablokowanie możliwości weryfikacji zaprogramowanej pamięci programu.

Dlatego należy pamiętać, że bity zabezpieczające programuje się w zależności od potrzeb, ale zawsze na końcu całego procesu programowania!

W układach z 89C51/C52 możliwe jest elektryczne kasowanie całej zawartości wewnętrznej pamięci programu poprzez podanie 10 milisekundowego impulsu programującego przy pozostałych sygnałach CF1...CF4 ustawionych jak podano w tabeli 3.

W wypadku układów z EPROM kasowanie pamięci może być wykonane tylko za pośrednictwem promieni ultrafioletowych (podobnie jak w typowych kostkach EPROM) o długości fali najlepiej około 4000Å (400 nm).

Do kasowania układów najlepiej jest stosować fabryczne kasownika EPROM lub samodzielnie wykonaną lampę kasującą np. ze świetlówki pracującej w podanym zakresie fal.

W tym miejscu chcę przestrzec niektórych z Was o możliwości zastosowania w roli lampy kasującej występujących na naszym rynku małych świetlówek do tzw. „sprawdzania banknotów”. Niestety nie nadają się one do kasowania struktur EPROM ze względu na nieodpowiednie widmo promieniowania.

Trzeba zatem nabyć specjalną świetlówkę, najlepiej miniaturową (o mocy 4...8W) emitującą stosowne promieniowanie. Cechą charakterystyczną właściwej dla naszych celów świetlówki jest całkowita przezroczystość rurki — czyli brak luminoforu. Jeżeli natraficie w sklepie na taki egzemplarz, pracujący w zakresie ultrafioletu, to z pewnością kasowanie układów za pomocą takiej lampy będzie udane.

Pamiętajcie tylko aby własnoręcznie wykonany „kasownik” zamknąć w obudowie, w przeciwnym przypadku ostre promieniowanie ultrafioletowe może uszkodzić wzrok!

Jeżeli ktoś chciałby spróbować samodzielnie wykonać chociażby najprostszy programator i zaprogramować procesor na podstawie niniejszego artykułu, zalecam sięgnięcie do literatury [1] i [2]. Uprzedzam jednak że ze względu na różnorodność parametrów czasowych w przebiegach z rys.3 i 4 w zależności od producenta układu oraz jego wersji, nie jest to zadanie łatwe, a przynajmniej nie da się zrobić na kolanie. Można bowiem drogi często procesor po prostu uszkodzić. Dlatego jeszcze raz zalecam korzystanie z gotowych programatorów lub złożenie samodzielnie proponowanego w artykule programatora AVT.

Sławomir Surowiński

Literatura:

- [1] – 80C51 Based 8-bit Microcontrollers, katalog Philips IC20
- [2] — Microcontrollers DataBook, Atmel 1995/97



Po dłuższej przerwie, spowodowanej po części wakacjami, zamieszczam dalsze komentarze dotyczące waszych listów, w których poruszacie problematykę opisywaną w serii artykułów o 8051. Dziś kolejna porcja listów oraz dodatkowe sprostowania błędów, które wkrały się do druku podczas tworzenia artykułów klasy mikroprocesorowej. Przedstawiam również kilka nadesłanych rozwiązań zadania dotyczącego zegara czasu rzeczywistego. Na koniec znajdziecie prawdziwy kasek dla bardziej zaawansowanych studentów klasy mikroprocesorowej mowa będzie mianowicie o dwóch aplikacjach na komputer edukacyjny, przysłanych do redakcji przez jednego z Czytelników. Jedna z nich pozwoli na programowanie pamięci EEPROM, a druga na programowanie mikrokontrolera 89C2051!



## 8051 Errare Humanum Est



### LIST 1

**Zenon Rakoczy** z Chropaczowa słusznie zwrócił uwagę na błąd, który wkrał się w program monitora komputerka edukacyjnego, zawartego w pamięci EPROM. Chodzi bowiem o błędne obliczanie sumy kontrolnej w procedurze monitora „SAVE”, która dostępna jest pod klawiszem „8” klawiatury komputerka.

W wyniku tego plik utworzony w formacie Intel-HEX z zawartością pamięci RAM komputerka jest nieprawidłowy, a w zasadzie nieprawidłowa jest tylko suma, będąca ostatnim bajtem w każdej linii tego zbioru. O formacie Intel-HEX pisałem w zeszłym roku na łamach bratniego pisma - Elektroniki Praktycznej.

W wyniku braku instrukcji

```
CLR C
```

w procedurze, przed obliczeniem wspomnianej sumy kontrolnej podczas odejmowania (instrukcja SUBB ...) następuje także niezamierzone odjęcie znacznika przeniesienia C. Kiedy znacznik ten jest równy 0, wszystko jest w porządku, jednak kiedy równa się „1” wtedy obliczona suma kontrolna jest mniejsza o 1 od prawidłowej. Stąd po powtórnym załadowaniu tak utworzonego zbioru Intel-HEX, do komputerka (komenda „LOAD” monitora), wystąpi komunikat „Err”, co świadczy o błędzie w linii zbioru.

Przyjrzyjmy się temu bliżej. Otóż w programie monitora obliczanie sumy kontrolnej odbywa się następująco:

```
.....
..... ;w zmiennej CRC znajduje się suma (mod 100h)
 ;wszystkich bajtów rekordu (linii) danych
 ;teraz nastąpi obliczenie sumy kontrolnej
CLR A ;wyzierowanie akumulatora
SUBB A, CRC ;obliczenie sumy wg wzoru: suma = 100h - CRC
 ;ale ze względu na instrukcję SUBB tak naprawdę
LCALL SEND ;wykonywane jest działanie: suma = 100h - CRC - C
..... ;wysłanie sumy przez port szeregowy
```

Jak z tego widać w przypadku gdy znacznik C jest równy „1” suma kontrolna obliczona zostanie nieprawidłowo wg wzoru:

suma = 100h - CRC - 1,

czyli będzie o 1 mniejsza od prawidłowej. Dlatego przed instrukcją odejmowania SUBB należało dodać instrukcję kasującą przypadkowo ustawiony znacznik C.

```
CLR A
CLR C ;brakująca instrukcja
SUBB A, CRC
LCALL SEND ;wysłanie akumulatora - sumy
```

Prawidłową sumę kontrolną można także uzyskać stosując negację logiczną z inkrementacją otrzymanej sumy bajtów rekordu, oto instrukcje:

```
MOV A, CRC
CPL C
INC A
LCALL SEND ;wysłanie akumulatora - sumy
```

W dalszej części opisu będę posługiwał się tą metodą obliczenia sumy kontrolnej rekordu danych w zbiorze Intel-HEX.

W związku z tym, że błąd został poprawiony w oprogramowaniu monitora na przełomie maja i czerwca, nabywcy zestawów AVT-2250 z okresu przed tą datą mogą mieć problemy z prawidłowym generowaniem zbiorów HEX przez komputer. Zenon Rakoczy pyta jak rozwiązać ten problem. Sposobów jest kilka.

### ROZWIĄZANIE NR 1

Poniżej zamieszczony jest listing 1 prawidłowej procedury „SAVE”, którą można skompilować z dowolnym przesunięciem względem początku zewnętrznej pamięci operacyjnej komputerka, a następnie używać, ładując ją w razie potrzeby tak jak każdy inny program tworzony przez siebie (uruchamiając ją poleceniem „JUMP”).

Pogrubioną czcionką zaznaczono linie programu, które obliczają ostatni bajt transmitowanego rekordu danych. W tej procedurze rejestr R6 wykorzystywany jest jako zmienna CRC, o której mówiłem wcześniej. W procedurze wykorzystano dwie dodatkowe podprocedury umieszczone w ciele monitora, a mianowicie „NULLKEY” (adres: 02CDh) oraz procedurę RSTEXT (adres 02DEh) zawarte w pamięci EPROM monitora. Zadaniem pierwszej procedury jest oczekiwanie na zwolnienie klawisza klawiatury, druga procedura przesyła ciąg znaków ASCII tzw. „string” poprzez złącze portu szeregowego komputerka, którego adres jest podany w rejestrze DPTR.

## Też to potrafisz

### Listing 1

```
1 CPU '8052.def'
2 include 'const.inc'
3 include 'bios.inc'
4 ;*****
5 ;Poprawiona procedura wysylania zawartosci RAM
6 ;komputerka w formacie Intel-HEX
7 ;wersja relokowalna od adresu 8000h (w ext.RAM)
8 ;*****
9 8000 org 8000h
10 ;*****
11 8000 SAVDATA:
12 8000 75786D mov DL1,#_5
13 8003 1202CD lcall 02CDh ;adres procedury NULLKEY
14 8006 C2D5 clr F0 ;flaga dot. konca
15 8008 75F005 mov B,#5
16 800B 7480 mov A,#80h ;#nullkey
17 800D 120295 lcall DELAY
18
19 8010 1203B9 lcall GETDPTR
20 8013 C083 push DPH
21 8015 C082 push DPL ;zachowaj na stosie
22 8017 75F005 mov B,#5
23 801A 1203B9 lcall GETDPTR
24 801D A3 inc DPTR
25 801E 858304 mov 04,DPH
26 8021 858205 mov 05,DPL
27 8024 D003 pop 03
28 8026 D002 pop 02
29 8028 7E00 mov R6,#0 ;CRC=0
30
31 802A 1202C5 qwer: lcall CONIN
32 802D B40DFA cjne A,#klaw_OK,qwer
33
34 8030 8A83 xnowy: mov DPH,R2
35 8032 8B82 mov DPL,R3
36 8034 75F005 mov B,#5
37 8037 12025F lcall DPTR4HEX ;wypisanie adresu
38
39 803A 743A mov A,#':'
40 803C 1202B9 lcall OUTRS
41 803F 7F10 mov R7,#10h ;16 bajtow danych
42 8041 EF mov A,R7
43 8042 FE mov R6,A
44 8043 128090 lcall send ;CRC = liczba bajtow w rekordzie
45 8046 EA mov A,R2 ;wyslanie liczby bajtów
46 8047 128090 lcall send ;zaladowanie adresu MSB
47 804A EE mov A,R6 ;i wyslanie go
48 804B 2A add A,R2
49 804C FE mov R6,A ;CRC = CRC + MSB adresu
50 804D EB mov A,R3 ;zaladowanie adresu LSB
51 804E 128090 lcall send ;i wyslanie go
52 8051 EE mov A,R6
53 8052 2B add A,R3
54 8053 FE mov R6,A ;CRC = CRC + LSB adresu
55 8054 E4 clr A
56 8055 128090 lcall send ;wyslanie: 00 - oznacza dane
57 8058 8A83 mov DPH,R2
58 805A 8B82 mov DPL,R3
59
60 805C E0 xznów: movx A,@DPTR ;pobranie danej z komorki pamieci RAM
61 805D 128090 lcall send ;i wyslanie jej
62 8060 E0 movx A,@DPTR ;ponowne pobranie tej danej
63 8061 2E add A,R6 ;CRC = CRC + dana
64 8062 FE mov R6,A
65 8063 A3 inc DPTR ;zwiększenie adresu
66
67 8064 AA83 niewp: mov R2,DPH ;i zapamietanie w R2.R3
68 8066 AB82 mov R3,DPL
69
70 8068 EA mov A,R2 ;porownanie MSB adresu pocz. i konca
71 8069 6C xrl A,R4 ;czy takie same
72 806A 7006 jnz dlej ;nie to skocz dalej (do etyk.'dlej')
73 806C EB mov A,R3 ;porownanie LSB adresu pocz. i konca
74 806D 6D xrl A,R5 ;czy takie same
75 806E 7002 jnz dlej ;nie to skocz dalej (do etyk.'dlej')
76 8070 D2D5 setb F0 ;flaga ozn. koniec obszaru
77
78 8072 DFE8 dlej: djnz R7,xznów ;następny bajt danej z RAM
79 8074 EE mov A,R6 ;zaladowanie CRC
80 8075 F4 cpl A ;i obliczenie reszty wg. wzoru
81 8076 04 inc A ;reszta = 100h - CRC
82 8077 128090 lcall send ;i wyslanie jej
83
84 807A 740D mov A,#0Dh ;wyslanie...
85 807C 1202B9 lcall OUTRS ;konca...
```

## Listing 1, cd.

```

86 807F 740A mov A,#0Ah ;wiersza, znaki #13#10...
87 8081 1202B9 lcall OUTRS ;tzw. eoln
88 8084 30D5A9 jnb F0,xnowy ;jesli nie koniec to nowy rekord
89
90 8087 C2D5 clr F0 ;jesli koniec to...
91 8089 90809E mov DPTR,#rs_end ;wyslanie konca pliku ':00000001FF'
92 808C 1202DE lcall 02DEh ;za pomoca dodatkowej procedury monitora
RSTEXT
93 808F 22 ret ;i koniec podprogramu
94
95 8090 120235 send: lcall HEXASCII ;procedura dodatkowa SEND:
96 8093 C5F0 xch A,B ;wysyla bajt jako 2 znaki ASCII
97 8095 1202B9 lcall OUTRS
98 8098 E5F0 mov A,B
99 809A 1202B9 lcall OUTRS
100 809D 22 ret
101
102 809E 3A303030 80A2 30303030 80A6 3146460D 80AA 0A00
;rs_end db ':00000001FF',13,10,0
;*****
103
104 80AC END

```

Kompilacja zakonczona pomyslnie!  
Zbior: „sav8000.s03”, 172 bajt(ow), 0.3 sekund(y).

## ROZWIĄZANIE NR 2

Innym sposobem ominięcia tej trudności jest własnoręczne zmodyfikowanie programu monitora umieszczonego w pamięci EPROM. W tym jednak przypadku niezbędny jest dostęp do programatora pamięci EPROM lub posiadanie takiego urządzenia. Kroki, jakie trzeba podjąć w tym przypadku, pozwolą na modyfikację monitora w taki sposób, że naciśnięcie klawisza „8” wywoła potem prawidłową procedurę wysyłania zawartości pamięci RAM poprzez łącze szeregowe komputerka. Oto one.

1. Odczytać programatorem zawartość EPROM komputerka 27C64
2. Korzystając z opcji „Edit” programatora zmodyfikować następujące komórki pamięci spod podanych niżej adresów:

| Adres: | Jest:    | Ma być:  |
|--------|----------|----------|
| 0701h  | 91 A9    | F1 21    |
| 0721h  | 43 71 24 | 12 09 00 |

3. Od adresu 0900h załadować do bufora programatora ww. kod programu zmodyfikowanej procedury SAVE (listing 1) skompilowany od tego adresu, za pomocą dyrektywy kompilatora

ORG 0900h ;zamiast ORG 8000h pozostawiając wcześniejszy – odczytany z EPROM kod programu monitora, bez zmian. Można także zamiast tego przepisać niżej wymienione dane z adresu 0900h do bufora programatora – listing 2.

Łatwiej jest jednak skorzystać z programu kompilatora PASM51.EXE i skompilować listing do postaci akceptowanej przez program obsługi programatora, czyli np. za pomocą instrukcji:

PASM51.EXE <zbiór> /h {Enter}

4. Zaprogramować ponownie pamięć EPROM tak zmodyfikowanym kodem monitora (wcześniej należy starą pamięć skasować promieniami ultrafioletowymi, lub posłużyć się inną, czystą kostką 27C64). Można także użyć pamięci EEPROM typu 28C64 (w dalszej części artykułu jeden z czytelników przedstawi prostą przystawkę do komputerka, służącą do programowania tej pamięci).

W ten sposób otrzymamy poprawiony program monitora, w którym procedura zapisu pamięci RAM komputerka SAVE będzie działać prawidłowo, a wywołanie jej będzie odbywać się tak jak poprzednio – za pomocą klawisza „8” klawiatury komputerka.

## Listing 2

```

Adres Dane

0900 75 78 6D0903 12 02 CD C2 D5 75 F0 05 74 80 12 02 95 12 03 B90913
C0 83 C0 82 75 F0 05 12 03 B9 A3 85 83 04 85 820923 05 D0 03 D0 02 7E 00
12 02 C5 B4 0D FA 8A 83 8B0933 82 75 F0 05 12 02 5F 74 3A 12 02 B9 7F 10
EF FE0943 12 09 90 EA 12 09 90 EE 2A FE EB 12 09 90 EE 2B
0953 FE E4 12 09 90 8A 83 8B 82 E0 12 09 90 E0 2E FE
0963 A3 AA 83 AB 82 EA 6C 70 06 EB 6D 70 02 D2 D5 DF
0973 E8 EE F4 04 12 09 90 74 0D 12 02 B9 74 0A 12 02
0983 B9 30 D5 A9 C2 D5 90 09 9E 12 02 DE 22 12 02 35
0993 C5 F0 12 02 B9 E5 F0 12 02 B9 22 3A 30 30 30 30
09A3 30 30 30 31 46 46 0D 0A 00

```

Podziękowania kieruję także w stronę p. Marka Lewandowskiego, od którego otrzymałem pocztą e-mailową informację o przedstawionym tu błędzie.



## LIST 2

**Zbigniew Wawryń** z Wołowa w swoim liście słusznie uważał pewną nieścisłość w wyjaśnieniach dotyczących obsługi bufora portu szeregowego – rejestru SBUF, o którym pisałem w kwietniowym numerze EdW na stronie 35 (górna prawa szpalta). Otóż stwierdzenie, że „... rejestr SBUF można adresować także za pomocą metody pośredniej (poprzez rejestr wskaźnikowy @Ri)...” jest oczywiście błędne, bowiem instrukcje operujące na tym wskaźniku z adresem powyżej 7Fh będą oczywiście operowały na górnej części pamięci RAM procesora, dostępnej tylko w kostkach 80C52 i pochodnych (87C52). Jak zapewne przypominacie sobie, w procesorach tych, w odróżnieniu od 80C51, znajduje się dodatkowe 128 bajtów wewnętrznej pamięci RAM umieszczonych pod adresami 80h...FFh. Dostęp do tych komórek jest jednak możliwy tylko za pomocą właśnie adresowania bezpośredniego, czyli poprzez rejestry R0 i R1 przy użyciu instrukcji np.

```
MOV R1, #90h ; odczytaj z komórki o adresie 90h
MOV A,@R1 ; daną i umieść w akumulatorze
```

W przypadku chęci zaadresowania tej komórki za pomocą instrukcji np.

```
MOV A, 90h
```

do akumulatora zostanie załadowana zawartość rejestru 90h umieszczonego w obszarze SFR procesora, czyli w tym wypadku rejestr portu P1, a nie komórka dodatkowej pamięci RAM. Do adresowania rejestru SBUF należy oczywiście użyć adresowania bezpośredniego, czyli instrukcji np.

```
MOV A, SBUF ;załadowanie odebranego znaku
 do akumulatora
```

## Też to potrafisz

Przyznam, że nie wiem co skłoniło mnie do przedstawienia takiego wywodu, bo w kilkuletniej praktyce nigdy nie zastosowałem błędnego adresowania.

Druga uwaga dotyczy niejasnego wywodu dotyczącego omawiania rejestrów odpowiedzialnych za przerwania, zamieszczonego w EdW nr 5/98 str. 38, prawa górna szpalta. Prawdą jest, że priorytet przerwań jest ustawiony fabrycznie i tak np. najwyższy priorytet ma przerwanie od wejścia INTO. „...Niestety dalsze stwierdzenie, że przerwanie jedno zostanie przerwane przez drugie dlatego, że jest dalsze w kolejności fabrycznie ustalonej, jest błędne. Szywny priorytet przerwań ma zastosowanie jedynie do rozstrzygnięcia kolejności przerwań jednocześnie nadchodzących. Natomiast do ustalania, które przerwanie może być przerwane przez inne służy tylko rejestr priorytetu IP...” – koniec cytatu p. Zbigniewa.

Zbigniewie, przeczytawszy kilka razy ten fragment artykułu muszę przyznać, że opisując ten problem za pomocą języka prostego „do granic możliwości”, być może sam ugryzłem się w język i ... zamiast napisać to co miałem na myśli, przełamałem tę myśl w sposób niewłaściwy. W każdym razie, obydwu nam chodzi o to samo. Ci spośród Czytelników, którzy niewłaściwie, zrozumieli wyjaśnienia proszeni są o zweryfikowanie swoich wiadomości. Postaram się pewnie dość skomplikowane dla Was, a jednocześnie oczywiste dla mnie problemy „8051”, opisywać bardziej jasno, tak aby nie było wątpliwości.



### LIST 3

Pan podpisujący się jako **RYŚ** lat XX, pyta w swoim liście, dlaczego w listingu programu z lekcji 8 str. 46 szpalta 2, linia 74 używam instrukcji

ORL TMOD, #00h

która przecież nie zmienia zawartości rejestru TMOD. I faktycznie instrukcja nie wpływa na zawartość rejestru, bowiem jest to logiczne dodać liczbę „0” do rejestru TMOD. W programie instrukcja ta jednak się pojawia, bowiem w praktyce podczas pisania innych aplikacji wykorzystujących np. licznik T1 (lub inny) występuje potrzeba modyfikacji czwórki rejestru TMOD (starszych 4 bitów dla licznika T1, młodszych 4 bitów dla licznika T0) i najpierw należy wyzerować tę czwórkę, którą chcemy zmienić instrukcją ANL, a następnie ustawiamy bity tej czwórki właśnie za pomocą instrukcji ORL. W przykładzie podanym w artykule, do wymaganej pracy licznika T1 nie potrzeba ustawiania żadnego z 4 starszych bitów rejestru TMOD, ale instrukcję w linii 74 wstawiłem domyślnie w celu ukazania, że w innym przypadku wystarczy jej użyć jedynie ze zmodyfikowanym drugim jej argumentem, tak aby uzyskać zamierzony efekt.

Tak więc zastosowanie tej instrukcji ma tylko i wyłącznie znaczenie poznawcze i oczywiście nie wpływa w tym przypadku na zawartość rejestru TMOD.

Aby usunąć swoje wątpliwości, Pan Ryś, samodzielnie napisał program do testowania operacji logicznych, które wykonuje procesor 8051. Gratuluję świetnego pomysłu. Życzę powodzenia przy pisaniu kolejnych programów.



### LIST 4

**Tomasz Kutyla** ze Stalowej Woli zauważył nieścisłości w druk dotyczący obliczeń liczników procesora, zamieszczonych w numerze EdW 5/98 (ach, to chyba naprawdę pechowcy numer...!). I tak:

- na str. 38, szpalta 2, pod rysunkiem rejestru IP w opisie jego bitów dwukrotnie powtórzona została para bitów PX1 i PT1;
- w lekcji nr 8, str. 45, szpalta 1, wiersz 17 (od dołu) jest  $fz = Fxtal / 12 / 32 = 28000 \text{ Hz}$   
a powinno być oczywiście ...  
 $28800 \text{ Hz}$ !
- na tej samej stronie i kolumnie, wiersz 6 (od dołu) wydrukowano:  
 $fz / 128 = 28800 / 256 = 225 (=THimp)$   
a powinno być oczywiście:  
 $fz / 128 = 28800 / 128 = 225 \dots \text{ itd.}$
- na stronie 45, szpalta 2, wiersz 4 (od góry) wydrukowano:  
 $TH1pocz = TH1max - TH1imp + 1 = 256 - 225 + 1 = 31$   
a powinno być oczywiście napisane:  
 $TH1pocz = TH1max - TH1imp + 1 = 255 - 225 + 1 = 31$
- na stronie 46 w listingu programu po linii 71 – z etykietą START, zabrakło przy okazji inicjacji układu przerwań, instrukcji  
SETB EA ; uaktywnienie systemu przerwań

Program będzie oczywiście działał na komputerku edukacyjnym AVT-2250, bowiem sam monitor komputerka wcześniej wykonuje tę instrukcję i uruchamia system przerwań. Jednak przy pisaniu programów na samodzielne systemy z procesorem 8051 lub podobnymi, nie moż-

na zapomnieć o globalnym ich odblokowaniu, po odpowiednim ustawieniu bitów w rejestrze masek przerwań. W przeciwnym wypadku układ będzie przysłowiowo „martwy”.

Tak więc wspomniana wcześniej instrukcja powinna znaleźć się dla porządku po linii 79, kiedy to ustawione zostały bity maskujące przerwanie od licznika T1 i rejestru priorytetu przerwań.

Tomaszu, dziękuję za te uwagi i ... gratuluję wnikliwej lektury naszych artykułów oraz „dobrego oka”.



### LIST 5

**Marcin Wiązania** z Kieleckiego zauważył nieścisłość w lekcji 8 z nr 5/98 EdW str. 44, szpalta 1, wiersz 30 (od dołu). Otóż stwierdzam tam, niesłusznie zresztą, że „... w przypadku wartości rezonatora 11059200 Hz zliczanie 1/100 sekundy byłoby dość kłopotliwe ze względu na to, że wartość tego kwarcu nie dzieli się przez 12 i dodatkowo przez liczbę całkowitą, tak aby dać liczbę 100...”. Otóż dzieli się - a tą liczbą, jak słusznie zauważa Marcin, jest przecież: 9261, bo:

$$11059200 / 12 = 921600,$$

$$921600 / 100 = 9216 - \text{szukana liczba całkowita}$$

Stąd wartość początkową licznika T1 można obliczyć następująco:

$$T1pocz = TH1.TL1 = 65535 - 9216 + 1 = 56320 = DC00h$$

Przy tej wartości początkowej wpisywanej do licznika za pomocą instrukcji np.

MOV TH1, #0DCh

; olr TL1, #0 nie jest konieczne!

licznik T1 będzie przepełniany dokładnie 100 razy na sekundę, czyli tak jak chcieliśmy.

Pozdrawiam Pana, panie Marcinie.

## Rozwiązanie zadania „ZEGAR”



### LIST 6

To tyle uwag naszych Czytelników, co do treści lekcji nr 8. W dalszej części artykułu przedstawię najciekawsze rozwiązania zadania z lekcji nr 8, a mianowicie wyposażenie zegara czasu rzeczywistego w dodatkową funkcję wyświetlania daty.

Na szczególną uwagę zasługują rozwiązania dwóch naszych czytelników. Pierwszy list otrzymałem od p. **Marcina Wiązania** z woj. kieleckiego (**listing 3**). Jego program zegara z datownikiem oceniam na 5+, autor bowiem podjął się także kontroli i prawidłowego wyświetlania liczby dni dla każdego miesiąca. Wiemy przecież, że liczba dni różni się w każdym miesiącu, a lutym różnica ta wynosi nawet 3 dni!

Oto fragment listu p. Marcina:

„... Problem ilości dni w każdym miesiącu rozwiązałem za pomocą tabeli stałych z wartościami dni w każdym miesiącu. Jest to rozwiązanie bardzo proste, ale nie uwzględniające lat przestępnych i nie można go zastosować kiedy ograniczona jest ilość pamięci programu...” (to akurat nie jest problemem – przyp. redakcji). „...Po uruchomieniu programu należy najpierw wprowadzić datę w następującej kolejności „dzień” : „miesiąc” : „rok”, a następnie godzinę...” (godzinę, minuty i sekundy – przyp. redakcji). „Przełączenia na wyświetlanie daty dokonuje się za pomocą klawisza „1”, natomiast z powrotem na godzinę, za pomocą klawisza OK. Klawisz sterujący możemy bardzo łatwo zmienić wprowadzając wartość innego klawisza. Do odczytu klawisza wykorzystałem zmienną „klawisz”. Po wprowadzeniu daty, program zgodnie z wprowadzonym miesiacem odczytuje z „tabeli stałych” ilość dni, a następnie umieszcza je w zmiennej „mies”. Zmienna „mies” określa w procedurze przerwania ilość dni w danym miesiącu. Próbowalem także, aby data i godzina zmieniała się na przemian po naciśnięciu za każdym razem tego samego przycisku. Lecz nie udało mi się tego osiągnąć nawet po wprowadzeniu małego opóźnienia. Zbyt duże powodowało, że układ wcześniej wchodził do obsługi przerwania, przez co traciłem wartość akumulatora, natomiast zbyt mała wartość powodowała miganie wyświetlacza.

Specjalnie zamieściłem listing programu, a nie jego treść źródłową, bowiem dzięki temu pożytek mają z niego zarówno komputerowcy jak i „ręczniacy”.

Program p. Marcina działa bezbłędnie. Radzę zatem spróbować przetestować go na swoim sprzęcie w domowym zaciszu!



## Listing 3

```

1 CPU '8052.def' 2
;*****
3 ;Klasa mikroprocesorowa - LEKCJA 8
4 ;Program obsługi zegara czasu rzeczywistego
5 ;na komputer edukacyjny AVT-2250
6 ;*****
7 ;procedura korzysta z przerwania licznika T1 (tryb 0)
8 ;wykorzystywane sa 3 komórki wewn. RAM procesora
9 ;zegar liczy: godziny, minuty, sekundy, dni, miesiace i lata
10 ;w trybie 24-godzinnym
11 ;*****
12
13 include 'const.inc'
14 include 'bios.inc'
15
16 ;Definicje komorek w wewn. RAM procesora
17 ;zajmowane przez dane zegara
18
19 0060 GODZ equ 60h ;licznik godzin
20 0061 MIN equ 61h ;licznik minut
21 0062 SEK equ 62h ;licznik sekund
22 0064 DDequ 64h ;licznik dni
23 0065 MMequ 65h ;licznik miesiecy
24 0066 RRequ 66h ;licznik lat
25 0067 MIES equ 67h ;zmienna ilosci dni w miesiacu
26 0063 licz128 equ 63h ;licznik 1/128 sek
27
28 ;Definicje stalych wykorzystywanych w programie
29
30 001F Czest equ 31 ;wartosc pocz. licznika TH1
31
32 ;*****
33 ;Początek kodu programu
34 8000 org 8000h
35 8000 02807F jmp START ;petla glowna od etyk. START
36 ;*****
37 ;Wektor przerwania od licznika T1
38 801B org 801Bh
39 801B intT1:
40 801B 758D1F mov TH1,Czest ;początek proc. przer. T1
41 801E 0563 inc licz128 ;przeladowanie licznika T1
42 8020 E563 mov A,licz128 ;zwiększenie licznika 1/128sek.
43 8022 C2E7 clr Acc.7
44 8024 7052 jnz koniecT1
45 8026 E562 mov A,SEK
46 8028 2401 add A,#1 ;zwiększanie licznika sekund
47 802A D4 da A ;z korekcja dziesiętna
48 802B F562 mov SEK,A
49 802D B46048 cjne A,#60h,koniecT1 ;czy SEK > 59?, nie to skocz
50 8030 756200 mov SEK,#0 ;tak to wyzeruj sekundy i koryguj minuty
51 8033 E561 mov A,MIN
52 8035 2401 add A,#1 ;zwiększenie licznika minut
53 8037 D4 da A ;z korekcja dziesiętna
54 8038 F561 mov MIN,A
55 803A B4603B cjne A,#60h,koniecT1 ;czy MIN > 59?, nie to skocz
56 803D 756100 mov MIN,#0 ;tak to zeruj minuty i koryguj godziny
57 8040 E560 mov A,GODZ
58 8042 2401 add A,#1 ;zwiększenie licznika minut
59 8044 D4 da A ;z korekcja dziesiętna
60 8045 F560 mov GODZ,A
61 8047 B4242E cjne A,#24h,koniecT1 ;czy GODZ > 23?, nie to skoc
62 804A 756000 mov GODZ,#0 ;tak to zeruj godziny
63 804D E564 mov A,DD
64 804F 2401 add A,#1 ;zwiększenie licznika dni
65 8051 D4 da A ;z korekcja dziesiętna
66 8052 F564 mov DD,A
67 8054 B56721 cjne A,MIES,koniecT1 ;czy mies.>ilosc dni, nie to skocz
68 8057 756401 mov DD,#1 ;tak to koryguj dni
69 805A E565 mov A,MM
70 805C 2401 add A,#1 ;zwiększenie licznika miesiecy
71 805E C0E0 push A
72 8060 90815F mov DPTR,#tab_mies ;pobranie adresu tabeli ilosci dni w mies.
73 8063 93 movc A,@A+DPTR ;wpisanie do A ilosci dni w danym miesiacu
74 8064 F567 mov MIES,A ;a nastepnie przepisanie do zmiennej „MIES”
75 8066 D0E0 pop A
76 8068 D4 da A ;korekcja dziesiętna miesiecy
77 8069 F565 mov MM,A
78 806B B4130A cjne A,#13h,koniecT1 ;czy mies.>12, nie to skocz
79 806E 756501 mov MM,#1 ;tak to koryguj miesiace i rok
80 8071 E566 mov A,RR
81 8073 2401 add A,#1
82 8075 D4 da A ;korekcja dziesiętna roku
83 8076 F566 mov RR,A
84 8078 koniecT1:
85 8078 D082 pop DPL ;odtworzenie rejestrów

```

## Też to potrafisz

### Listing 3, cd.

```

86 807A D083 pop DPH ;ze stosu
87 807C D0E0 pop Acc
88 807E 32 reti
89
90 807F ;*****
91 807F C28E clr TR1 ;licznik T1 stop
92 8081 53890F anl TMOD,#0Fh ;wyczyszczenie bitow T1
93 8084 438900 orl TMOD,#00h ;T1 jako 16-bitowy (tryb 1)
94 8087 758D1F mov TH1,#Czest ;zaladowanie licznika
95 808A 756300 mov licz128,#0 ;wyzerowanie licznika 1/256sek.
96 808D 757280 mov intvec,#80h ;zaladowanie MSB wektora przerwan
97 8090 D2AB setb ET1 ;odblokowanie przerwania od T1
98 8092 D2BB setb PT1 ;priorytet na to przerwanie
99 8094
100 8094 120274 lcall CLS ;wyczyszczenie displeja
101 8097 757840 mov DL1,#_minus
102 809A 757940 mov DL2,#_minus
103 809D 75F001 mov B,#1
104 80A0 1203A7 lcall GETACC ;pobranie poczatkowych dni
105 80A3 F564 mov DD,A
106 80A5 757A40 mov DL3,#_minus
107 80A8 757B40 mov DL4,#_minus
108 80AB 757C40 mov DL5,#_minus
109 80AE 75F004 mov B,#4
110 80B1 1203A7 lcall GETACC ;pobranie poczatkowych miesiecy
111 80B4 F565 mov MM,A
112 80B6 757D40 mov DL6,#_minus
113 80B9 90815F mov DPTR,#tab_mies ;pobranie z tabeli ilosci dni zaleznej
114 80BC 93 movc A,@A+DPTR ;od wpisanego miesiaca
115 80BD F567 mov MIES,A
116 80BF 757E40 mov DL7,#_minus
117 80C2 757F40 mov DL8,#_minus
118 80C5 75F007 mov B,#7
119 80C8 1203A7 lcall GETACC ;pobranie poczatkowego roku
120 80CB F566 mov RR,A
121 80CD 120274 lcall CLS ;wyczyszczenie displeja
122 80D0 757840 mov DL1,#_minus
123 80D3 757940 mov DL2,#_minus
124 80D6 75F001 mov B,#1
125 80D9 1203A7 lcall GETACC ;pobranie poczatkowej godziny
126 80DC F560 mov GODZ,A
127 80DE 757B40 mov DL4,#_minus
128 80E1 757C40 mov DL5,#_minus
129 80E4 75F004 mov B,#4
130 80E7 1203A7 lcall GETACC ;pobranie poczatkowej minuty
131 80EA F561 mov MIN,A
132 80EC 757E40 mov DL7,#_minus
133 80EF 757F40 mov DL8,#_minus
134 80F2 75F007 mov B,#7
135 80F5 1203A7 lcall GETACC ;pobranie poczatkowej sekundy
136 80F8 F562 mov SEK,A
137 80FA 757A40 mov DL3,#_minus ;zapalenie kresk w postaci
138 80FD 757D40 mov DL6,#_minus ;GG-MM-SS (godzina wprowadzona!)
139 8100 74FA mov A,#250
140 8102 120295 lcall DELAY ;odczekanie ok. 0,5 sekundy
141 8105 1202C5 lcall CONIN ;czekanie na start zegara (klawisz)
142 8108 D28E setb TR1 ;start licznika (zegara)
143 810A pokaz:
144 810A E576 mov A,klawisz ;wpisanie zawartosci bufora klawisz do A
145 810C B43102 cjne A,#'1',zegar ;jezeli A rozne od 1 to skocz
146 810F 802D sjmp data ;a jezeli rowne, to kolejny rozkaz
147 8111 E563 zegar: mov A,licz128
148 8113 30E608 jnb Acc.6,pelne ;co 1/2 sekundy pokazuj na zmiane
149 8116 757A00 mov DL3,#0 ;puste DL3 i DL4
150 8119 757D00 mov DL6,#0
151 811C 8006 sjmp czas
152 811E 757A40 pelne: mov DL3,#_minus ;i kreski na DL3 i DL6
153 8121 757D40 mov DL6,#_minus
154 8124 czas:
155 8124 E560 mov A,GODZ
156 8126 75F001 mov B,#1 ;na DL1.DL2
157 8129 12024E lcall A2HEX ;wypisz godziny
158 812C E561 mov A,MIN
159 812E 75F004 mov B,#4 ;na DL4.DL5
160 8131 12024E lcall A2HEX ;wypisz minuty
161 8134 E562 mov A,SEK
162 8136 75F007 mov B,#7 ;na DL7.DL8
163 8139 12024E lcall A2HEX ;wypisz na wyswietlacz
164 813C 80CC sjmp pokaz ; i od poczatku
165 813E data:
166 813E E576 mov A,klawisz ;przepisanie bufora klawisz do A
167 8140 B40D02 cjne A,#klaw_OK,wypisz;jezeli A rozne od klaw_OK to skocz
168 8143 80C5 sjmp pokaz ;jezeli rowne, to wykon. kolejny rozkaz
169 8145 E564 wypisz: mov A,DD
170 8147 75F001 mov B,#1 ;na DL1.DL2

```

```

171 814A 12024E lcall A2HEX ;wypisz dni
172 814D E565 mov A,MM
173 814F 75F004 mov B,#4 ;na DL4.DL5
174 8152 12024E lcall A2HEX ;wypisz miesiace
175 8155 E566 mov A,RR
176 8157 75F007 mov B,#7 ;na DL7.DL8
177 815A 12024E lcall A2HEX ;wypisz na wyswietlacz rok
178 815D 80DF sjmp data ; i od poczatku
179
180 815F 00322932 tab_mies db 00h,32h,29h,32h,31h,32h,31h ;tablica dni w
181 8166 32323100 8163 313231 db 32h,32h,31h,00h,00h,00h,00h
182 816A 000000 816D 00003231 db 00h,00h,32h,31h,32h
183 8172 8171 32 ;kolejnych miesiacach
184 8172 END

```

Kompilacja zakonczona pomyslnie!  
Zbior: „lekcja8a.s03”, 346 bajt(ow), 0.4 sekund(y).



### LIST 7

Na zakończenie list od **Piotra Konopko** z Łomży, który także przesyła rozwiązanie problemu zadania z lekcji 8 oraz dodatkowy listing wersji zegara wyświetlającego setne części sekundy. Piotr pisze...

„... mam 19 lat i skończyłem właśnie technikum elektro-  
niczne i szykuję się na studia. Chciałbym serdecznie Panu  
podziękować za prowadzony na łamach EdW kurs progra-  
mowania 8051. Do niedawna byłem, jak to się mówi, zielo-  
ny na temat mikrokontrolerów, ale dzięki Panu i EdW szybko  
przyswoiłem podstawy 8051. Z lekcji na lekcję odnoszę co-  
raz większe sukcesy. Muszę Panu także powiedzieć, że wie-  
dza, jaką zdobyłem na kursie, szybko przyniosła mi sukces  
w szkole. Na zajęciach z programowania sterowników mik-  
roprocesorowych nie miałem żadnych problemów. Mimo  
nieco innych rozkazów i budowy sterowników założenia są  
takie same...” (ciekaw jestem jakiego rodzaju sterowniki  
Pan programuje w szkole? – przypis redakcji). „Dzięki temu,  
że poznałem zasady działania 8051, czułem tzw. „bluesa”.  
Wielka uniwersalność mikrokontrolerów, o jakiej pisał Pan  
na początku klasy mikroprocesorowej, w moim przypadku  
dała o sobie znać bardzo szybko. (...)”

„Na zakończenie chciałbym stwierdzić, że klasa prowadzo-  
na jest perfekcyjnie. Żaden belfer w dotychczasowej mojej  
edukacji nie uczył mnie tak szybko i sprawnie jak Pan. Jesz-  
cze raz Panu i całej redakcji EdW serdecznie dziękuję...”

Bardzo dziękuję za te ciepłe słowa, szczególnie po lektu-  
rze pechowego odcinka klasy mikroprocesorowej z nr 5/98.  
Jestem rad, że tak wielu z Was odnosi pierwsze sukcesy  
w programowaniu kontrolerów 8051 i to dzięki cyklowi  
moich artykułów w EdW.

W nawiązaniu do listu p. Piotra zamieszczam listing jego  
programu – zegara z wyświetlaniem setnych części sekun-  
dy, zgodnie z zasadami opisanymi przy okazji omawiania lis-  
tu 5. I chociaż program Piotra pracują znakomicie, to zwrac-  
am uwagę Panu i wszystkim początkującym programistom  
na dobrą zasadę umieszczania jak największej liczby komen-  
tarzy w programach źródłowych. Jak sami się bowiem prze-  
konacie, ich brak często utrudnia, lub wręcz uniemożliwia  
analizę programu (a nawet jego rozpoznanie) po pewnym  
czasie.

Oto listing programu zegara ze zliczaniem setnych części  
sekundy z kwarcem 11059200 Hz (listing 4).

Postarajcie się, Drodzy Czytelnicy, przeanalizować nie ko-  
mentowany program p. Piotra i uzupełnić go o komentarze.  
Z pewnością zrozumienie obcego kodu źródłowego, w do-  
datku bez komentarzy, będzie pouczającą lekcją dla każdego  
zapaleńca procesorów 8051 i nie tylko.

Na zakończenie korespondencji od p. Piotra Konopko  
przytaczam dodatkowy listing programu zamka szyfrowego  
(listing 5), który to p. Piotr wykonał samodzielnie jako swój  
pierwszy program na 8051. Pomimo braku komentarzy i z  
trochę zagmatwanej obsługi, program ten zasługuje na wy-  
różnienie i pochwałę, ze względu na swoją prostotę i funk-  
cjonalność, a przede wszystkim za to, że program „działa”  
i potrafi sterować dołączonym do portu P1.0 układem załą-  
czania rygla. Zanim przejdę do prezentacji listingu programu  
zamka szyfrowego, posłuchajmy opisu sposobu użytkowa-  
nia i działania programu. P. Piotr pisze w swoim liście:

### Listing 4

```

1 CPU 8052.def 2 include 'const.inc'
3 include 'bios.inc'
4
5 0060 godz equ 60h
6 0061 min equ 61h
7 0062 sek equ 62h
8 0063 liczl28 equ 63h
9 00DC czest equ DCh
10
11 8000 org 8000h
12 8000 028056 ljmp start
13 801B org 801Bh
14 801B inttl:
15 801B 758DDC mov TH1,#czest
16 801E E563 mov A,liczl28
17 8020 2401 add A,#1
18 8022 D4 da a
19 8023 F563 mov liczl28,a
20 8025 B40027 cjne A,#0h,koniec
21 8028 E562 mov A,sek
22 802A 2401 add A,#1
23 802C D4 da a
24 802D F562 mov sek,a
25 802F B4601D cjne A,#60h,koniec
26 8032 756200 mov sek,#0
27 8035 E561 mov A,min
28 8037 2401 add A,#1
29 8039 D4 da a
30 803A F561 mov min,a
31 803C B46010 cjne A,#60h,koniec
32 803F 756100 mov min,#0
33 8042 E560 mov A,godz
34 8044 2401 add A,#1
35 8046 D4 da a
36 8047 F560 mov godz,a
37 8049 B42403 cjne A,#24h,koniec
38 804C 756000 mov godz,#0
39 804F koniec:
40 804F D082 pop dpl
41 8051 D083 pop dph
42 8053 D0E0 pop acc
43 8055 32 reti
44
45 8056 start:
46 8056 C28E clr TR1
47 8058 53890F anl TMOD,#0fh
48 805B 438910 orl TMOD,#10h
49 805E 758DDC mov TH1,#czest
50 8061 756300 mov liczl28,#0
51 8064 757280 mov intvec,#80h
52 8067 D2AB setb ET1
53 8069 D2BB setb PT1
54 806B 120274 lcall CLS
55
56 806E 757840 mov DL1,#_minus
57 8071 757940 mov DL2,#_minus
58 8074 75F001 mov B,#1
59 8077 1203A7 lcall GETACC
60 807A F560 mov godz,a
61 807C 757A40 mov DL3,#_minus
62 807F 757B40 mov DL4,#_minus

```

## Też to potrafisz

„...Po załadowaniu programu pojawi się napis „HASLO”. Na początku brak jest wprowadzonego hasła, dlatego należy wcisnąć klawisz OK., potwierdzający. Pojawi się wtedy napis „YES”, świadczący o prawidłowym hasle. Następnie ponownie wciskamy klawisz OK. w celu wprowadzenia nowego hasła. Zostanie wyświetlony napis „NEU” (nowy) po którym to możemy wprowadzić nowy szyfr (liczba cyfr zależy od wielkości wolnych komórek w wewnętrznej RAM powyżej adresu 22h). Hasło potwierdzamy klawiszem OK. Teraz układ jest zaszyfrowany, a na wyświetlaczu widnieje napis „HASLO”. Aby teraz móc sterować wyjściem zamka (w moim przypadku jest to P1.0), należy podać prawidłowy szyfr. Pięciokrotne wprowadzenie błędnego hasła blokuje zamek, uniemożliwiając dalsze manipulacje. Żle wprowadzony kod sygnalizowany jest napisem „Error”. Po wprowadzeniu właściwego hasła zatwierdzamy je klawiszem OK. Wówczas mamy 2 możliwości: albo możemy odblokować wyjście (P1.0) naciskając dowolny klawisz oprócz OK. i M, lub możemy wprowadzić nowe hasło wciskając klawisz OK. Przez cały czas odblokowania zamka wyświetlany jest napis „YES”.

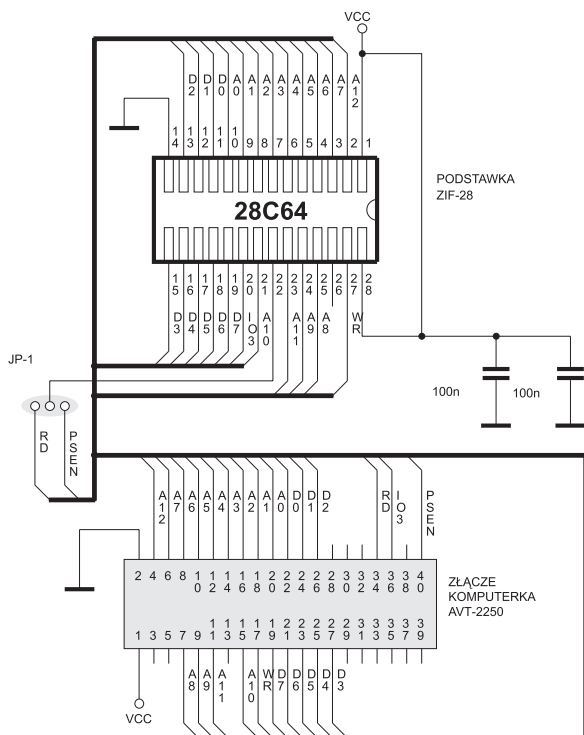
W celu ponownej aktywacji zamka wystarczy wcisnąć klawisz OK. Blokada automatycznie zostanie uaktywniona, a na wyświetlaczu ponownie zaświeci się napis HASLO...”

Brawo Piotrze!, życząc dalszych sukcesów i wielu ciekawych pomysłów na programy na 8051.

Na zakończenie naszej rubryki prezentuję prawdziwy „hit” tego numeru, mianowicie projekty: programatora pamięci EEPROM typu 28C64 (kompatybilna z 27C64, z tym że programowana i kasowana elektrycznie) oraz programatora mikroprocesora 89C2051 (który jest ograniczoną wersją procesora 87C51) w postaci przystawki do komputerka edukacyjnego AVT-2250. Projekt ten wykonał p. Tadeusz Kałuża z Podłęża (woj. krakowskie). Brawo panie Tadeuszu! Oto list:

„...Jestem 40-letnim rękodzielnikiem, zajmującym się amatorsko elektroniką. Analizując budowę komputerka edukacyjnego oraz uczestnicząc w prowadzonym przez Pana kursie programowania mikrokontrolerów jednoukładowych, doszedłem do wniosku, że funkcja edukacyjna dla komputerka to mało i wykonałem dwie proste przystawki, które jak uważam zdecydowanie zwiększają jego atrakcyjność. Pierwsza przystawka jest bardzo prosta i służy do programowania pamięci typu EEPROM np. 28C64. Przystawka ta składa się z zaciskowej podstawki podłączonej do łącza systemowego komputerka w sposób uwidoczniiony na schemacie (rysunek 1).

Rys. 1. Schemat przystawki programującej pamięć EEPROM 28C64



## Listing 4, cd.

```
63 8082 75F003 mov B,#3
64 8085 1203A7 lcall GETACC
65 8088 F561 mov min,a
66 808A 757C40 mov DL5,#_minus
67 808D 757D40 mov DL6,#_minus
68 8090 75F005 mov B,#5
69 8093 1203A7 lcall GETACC
70 8096 F562 mov sek,a
71 8098 74FA mov A,#250
72 809A 120295 lcall DELAY
73 809D 1202C5 lcall CONIN
74 80A0 D28E setb TR1
75 80A2 czas:
76 80A2 E560 mov A,godz
77 80A4 75F001 mov B,#1
78 80A7 12024E lcall A2HEX
79 80AA E561 mov A,min
80 80AC 75F003 mov B,#3
81 80AF 12024E lcall A2HEX
82 80B2 E562 mov A,sek
83 80B4 75F005 mov B,#5
84 80B7 12024E lcall A2HEX
85 80BA E563 mov A,licz128
86 80BC 75F007 mov B,#7
87 80BF 12024E lcall A2HEX
88 80C2 80DE sjmp czas
89
90 80C4 END
```

Kompilacja zakończona pomyslnie!  
Zbior: „zegar\_2.s03”, 172 bajt(ow), 0.3 sekund(y).

Kod programu wpisany do komputerka wygląda następująco:

```
START:
90 C0 00
E0
C0 83
53 83 9F
F0
D0 83
A3
74 02
12 02 95
75 F0 03
12 02 5F
74 E0
B5 83 E6
02 00 00
KONIEC
```

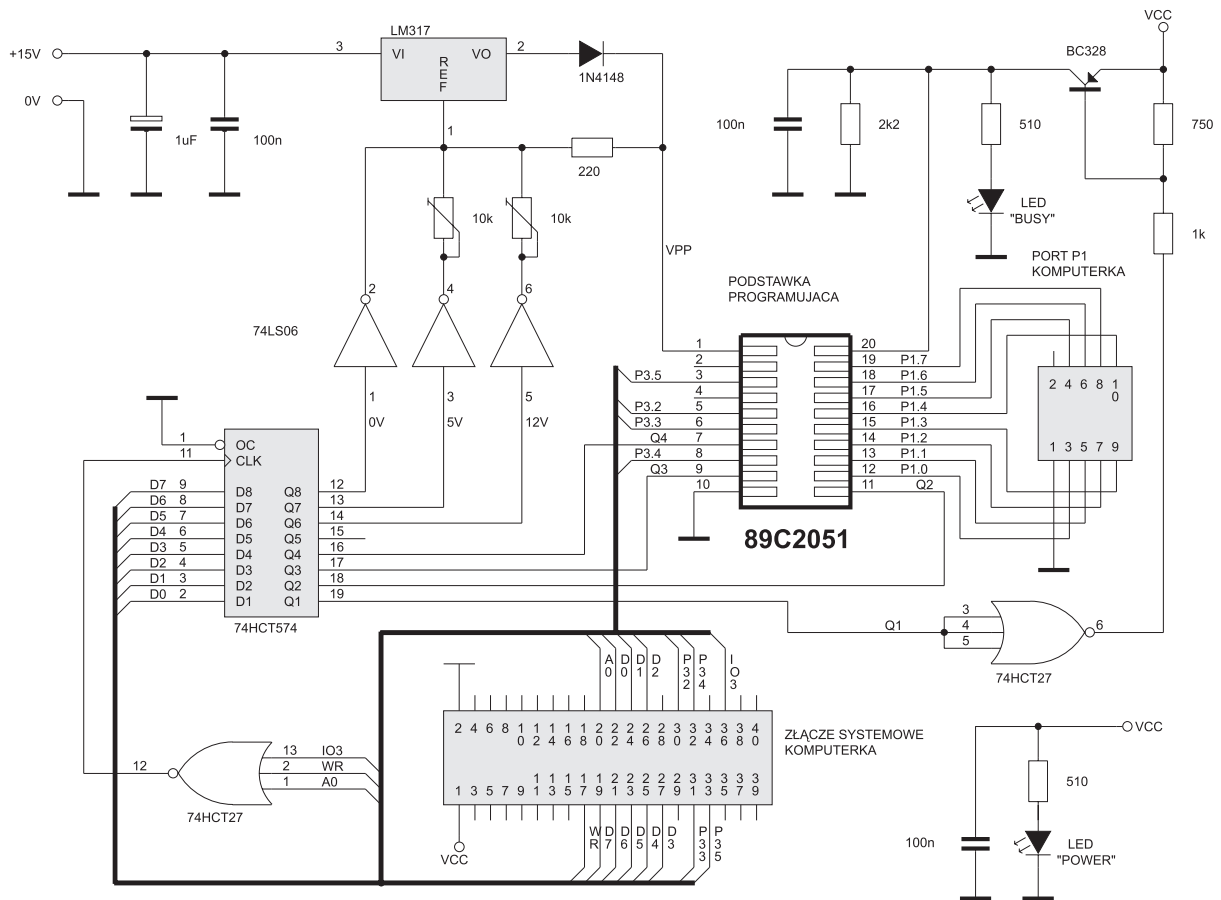
Po podłączeniu przystawki w komputerku JUMPER JP-3 należy ustawić na poz. C000h, a program wpisać w wolnym obszarze pamięci programu U3 lub w końcowym fragmencie aktywnego obszaru pamięci U4. Proces programowania rozpoczyna się wywołaniem wpisanego programu funkcją „JUMP” i powoduje przekopiowanie danych z obszaru adresowego C000h – DFFFh do programowanej pamięci, jednocześnie wyświetlając aktualny stan rejestru DPTR (czyli adresu aktualnie programowanej komórki – przyp. redakcji).

Następna przystawka jest bardziej złożona i służy do programowania mikrokontrolerów typu AT89C2051 firmy Atmel. Przy jej wykonaniu wzorowałem się na pańskich projektach AVT-320 i AVT-2250. Schemat tej przystawki przedstawia rysunek (rysunek 2).

Program uruchamiający przystawkę najlepiej wpisać w wolny obszar pamięci U3 komputerka (ostatecznie może być zapisany w pamięci U4 w obszarze adresowym D000h – DFFFh). Jumper JP-3 należy ustawić w pozycji C000h. Wywołanie programu funkcją „JUMP” powoduje ustawienie wszystkich istotnych w programowaniu wywołań podstawki w stan niski, a następnie oczekuje na naciśnięcie odpowiedniego klawisza w celu wywołania pożądanej procedury. W tym też czasie należy włożyć w podstawkę procesor 89C2051. Kod programu przedstawia listing 6.

Wywołanie programu powoduje wyświetlanie sekwencji – P1. 02. C3. i oczekuje na naciśnięcie następujących klawiszy:  
klawisz (1) – programowanie – odczyt, powoduje wpisanie do pamięci programowanego mikroprocesora danych umieszczonych w pamięci





Rys. 2. Schemat przystawki programującej procesor 89C2051 (89C1051)

### Listing 5

```

Listing 5
1
2 CPU 8052.def
3 include 'CONST.INC'
4 include 'BIOS.INC'
5 8000 org 8000h
6 8000 7A00 mov R2,#00h
7 8002 752000 mov 20h,#00h
8 8005 752100 mov 21h,#00h
9
10 8008 poczatek:
11 8008 120274 lcall cls
12 800B 75F002 mov B,#2
13 800E 9080E1 mov dptr,#tablica
14 8011 120285 lcall text
15 8014 7878 mov R0,#78h
16 8016 7922 mov R1,#22h
17 8018 skok10:
18 8018 1280DB lcall wait
19 801B 1202C5 lcall conin
20 801E E576 mov A,76h
21 8020 B40D37 cjne A,#13,skokpom2
22 8023 E520 mov A,20h
23 8025 6521 xrl A,21h
24 8027 704F jnz skok2
25 8029 120274 lcall cls
26 802C 757B6E mov dl4,#_Y
27 802F 757C79 mov dl5,#_E
28 8032 757D6D mov dl6,#_5
29 8035 7878 mov R0,#78h
30 8037 7922 mov R1,#22h
31 8039 1280DB lcall wait
32 803C 1202C5 lcall conin
33 803F E576 mov A,76h
34 8041 B40D55 cjne cjne
35 8044 757B54 mov dl4,#_n
36 8047 757C79 mov dl5,#_E
37 804A 757D3E mov dl6,#_U
38 804D skok5:
39 804D 1280DB lcall wait
40 8050 1202C5 lcall conin
41 8053 E576 mov A,76h
42 8055 B40D05 cjne A,#13,skok4
43 8058 80AE sjmp poczatek
44 805A skokpom2:
45 805A 0280B0 ljmp skok1
46 805D skok4:
47 805D F7 mov @R1,A
48 805E C000 push 00h
49 8060 E578 mov A,dll
50 8062 B40003 cjne A,#0,skok6
51 8065 120274 lcall cls
52 8068 skok6:
53 8068 D000 pop 0h
54 806A 09 inc R1
55 806B 0520 inc 20h
56 806D B88002 cjne R0,#80h,skok5
57 8070 80DB sjmp skok5
58 8072 skok9:
59 8072 7440 mov A,#_minus
60 8074 F6 mov @R0,A
61 8075 08 inc R0
62 8076 80D5 sjmp skok5
63 8078 skok2:
64 8078 120274 lcall cls
65 807B 757979 mov dl2,#_E
66 807E 757A50 mov dl3,#_r
67 8081 757B50 mov dl4,#_r
68 8084 757C3F mov dl5,#_0

```

## Też to potrafisz

### Listing 5, cd.

```

69 8087 757D50 mov dl6,#_r
70 808A 1280DB lcall wait
71 808D 1202C5 lcall conin
72 8090 0A inc R2
73 8091 752100 mov 21h,#00
74 8094 skok7:
75 8094 BA0550 cjne R2,#5,skokpom
76 8097 80FB sjmp skok7
77 8099 skok3:
78 8099 C290 clr 90h ;clr P1
79 809B 1280DB lcall wait
80 809E 1202C5 lcall conin
81 80A1 E576 mov A,76h
82 80A3 B40DF3 cjne A,#13,skok3
83 80A6 7A00 mov R2,#00h
84 80A8 752100 mov 21h,#00h
85 80AB D290 setb 90h
86 80AD 028008 ljmp poczatek
87 80B0 skok1:
88 80B0 67 xrl A,@R1
89 80B1 701C jnz skok8
90 80B3 0521 inc 21h
91 80B5 09 inc R1
92 80B6 B88003 cjne R0,#80h,skok11
93 80B9 028018 ljmp skok10
94 80BC skok11:
95 80BC C000 push 00h

96 80BE E578 mov A,d11
97 80C0 B40003 cjne A,#0,skok12
98 80C3 120274 lcall cls
99 80C6 skok12:
100 80C6 D000 pop 0h
101 80C8 7440 mov A,#_minus
102 80CA F6 mov @R0,a
103 80CB 08 inc R0
104 80CC 028018 ljmp skok10
105 80CF skok8:
106 80CF 740D mov A,#13
107 80D1 2521 add A,21h
108 80D3 F521 mov 21h,A
109 80D5 B880E4 cjne R0,#80h,skok11
110 80D8 028018 ljmp skok10
111 80DB wait:
112 80DB 74FF mov A,#255
113 80DD 120295 lcall delay
114 80E0 22 ret
115
116 80E1 76776D38 tablica db _H,_A,_5,_L,_0,0
117 80E5 3F00 skokpom:
118 80E7 ljmp poczatek
119
120 80EA END

Kompilacja zakonczona pomyslnie!
Zbior: „szyfr.s03”, 234 bajt(ow), 0.3 sekund(y).

```

U4 w obszarze adresowym C000h ... C7FFh oraz kontrolny odczyt do obszaru adresowego C800h ... CFFFh;

klawisz (2) – odczyt – kopiowanie pamięci programu mikroprocesora do obszaru adresowego C800h ... CFFFh w pamięci U4;

klawisz (3) – kasowanie pamięci programu mikroprocesora; zakończenie każdej procedury m sygnalizowane jest na wyświetlaczu komputerka i wtedy można wyjąć z podstawki mikroprocesor lub klawiszem (0) powrócić do miejsca wywołania procedur.

Opisane programy umieściłem w prosty sposób w pamięci U3 mojego komputerka, wykorzystując przystawkę do programowania pamięci 28C64 oraz prosty programik odczytu monitora.

Jeszcze raz dziękuję za wspaniałą zabawę, serdecznie pozdrawiam Pana i cały zespół redakcyjny EdW ...”.

Tym z Czytelników, którzy mają wątpliwości skąd wziąć te dane do programowania, wyjaśniam, że można je oczywiście załadować przed uruchomieniem programu przystawki programatora z komputera za po-

mocą instrukcji monitora „LOAD”, lub wpisać ręcznie (ręczniacy) za pomocą polecenia „EDIT” programu monitora.

Jeszcze raz brawa dla Pana, panie Tadeuszu, nie tylko za pomysł, ale i za wytrwałość w stworzeniu dość długiego, jak na ręczniaka, programu. Szkoda, że nie przysłał nam Pan komentarzy dotyczących swoich listingów. Proponuję Czytelnikom uzupełnienie tego samodzielnie i przeanalizowanie programu na poziomie mnemoników procesora 8051. Przekonajcie się, że proces tłumaczenia kodu maszynowego na źródłowy nie jest taki trudny. Tak a propos, to tłumaczenie nazywa się fachowo disasemblacją (operacja odwrotna do asemblacji, czyli tłumaczenia kodu źródłowego na maszynowy).

Wszystkim Autorom listów serdecznie dziękuję za wspaniałe pomysły i uwagi dotyczące moich artykułów, zaś pozostałym Czytelnikom życzę wielu ciekawych pomysłów. Piszcie do mnie!

**Sławomir Surowiński**

### Listing 6

```

START:
Adres Dane
x8B0 75 90 00 (*)
x8B3 C2 B2
x8B5 C2 B3
x8B7 C2 B4
x8B9 90 80 00
x8BC 74 80
x8BE F0
x8BF 12 02 74
x8C2 75 78 F3
x8C5 75 79 06
x8C8 75 7B BF
x8CB 75 7C 5B
x8CE 75 7E B9
x8D1 75 7F 4F
x8D4 12 02 C5
x8D7 B4 31 03
x8DA 02 x8 E9 (*)
x8DD B4 32 03
x8E0 02 x9 62 (*)
x8E3 B4 33 EE
x8E6 02 x9 70 (*)
PROGRAMOWANIE:
x8E9 12 02 74
x8EC 75 78 F5
x8EF 12 x9 AD (*)
x8F2 D2 B4
x8F4 74 47
x8F6 F0

x8F7 75 90 FF
x8FA 74 27
x8FC F0
x8FD 90 C0 00
x900 F5 90
x903 C2 B3
x905 00 00
x907 D2 B3
x909 A3
x90A 75 F0 05
x90D 12 02 5F
x910 74 02
x912 12 02 95
x915 D2 B2
x917 C2 B2
x919 74 C8
x91B B5 83 E2
x91E 90 80 00
x921 74 47
x923 F0
x924 75 90 FF
x927 90 C8 00
x92A C2 B4
x92C 00 00
x92E E5 90
x930 D2 B4
x932 F0
x933 75 F0 05
x936 12 02 5F
x939 D2 B2
x93B C2 B2
x93D A3

x93E 74 D0
x940 B5 83 07
x943 90 80 00
x946 74 87
x948 E0
x949 C2 B3
x94B C2 B4
x94D 75 90 00
x950 74 80
x952 F0
x953 75 79 3F
x956 75 7A DC
x959 12 02 C5
x95C B4 30 FA
x95F 02 x8 B0 (*)
(*) - „x” pierwsza cyfra
ustalonego adresu

ODCZYT:
x962 12 02 74
x965 75 79 3F
x968 12 x9 AD (*)
x96B D2 B4
x96D 02 x9 21 (*)
KASOWANIE:
x970 12 02 74
x973 75 78 B9
x976 12 x9 AD (*)
x979 74 49
x97B F0
x97C 75 90 FF

x97F 74 29
x981 F0
x982 00 00 00
x985 C2 B3
x987 74 0A
x989 12 02 95
x98C D2 B3
x98E 74 0A
x990 12 02 95
x993 74 49
x995 F0
x996 C2 B3
x998 74 81
x99A F0
x99B 75 90 00
x99E 74 80
x9A0 F0
x9A1 75 79 DC
x9A4 12 02 C5
x9A7 B4 30 FA
x9AA 02 x8 B0 (*)
x9AD 74 81
x9AF F0
x9B0 74 0A
x9B2 12 02 95
x9B5 74 41
x9B7 F0
x9B8 D2 B3
x9BA 22

KONIEC

```

Dzisiejszy odcinek cyklu o mikrokontrolerach rodziny 8051 poświęcimy układom peryferyjnym I/O, czyli tzw. wejścia-wyjścia (ang. "Input/Output").

Omówione zostaną typy układów, najciekawsze rozwiązania konstrukcyjne, od tych najprostszych do nieco bardziej skomplikowanych.

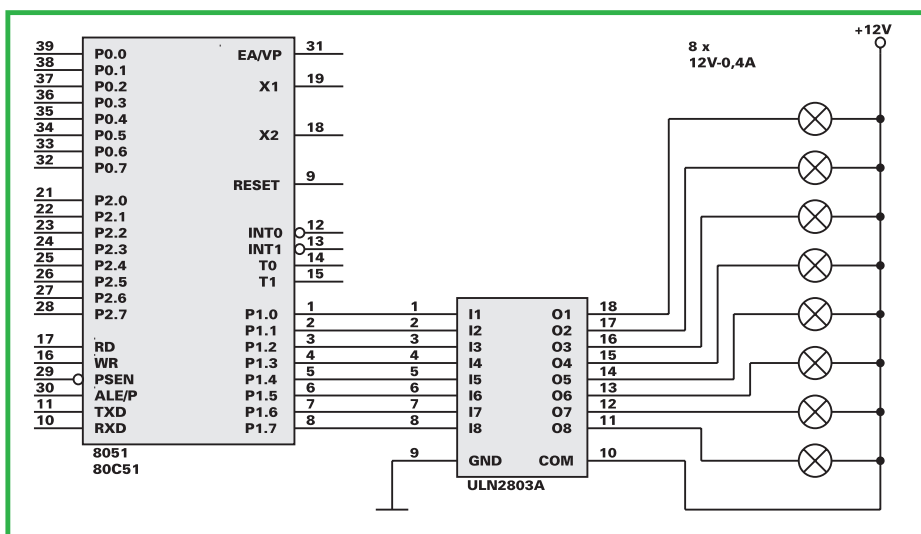
Aby jednak zachować związek z cyklem o mikroprocesorach, wszystkie przedstawione w artykule układy będą w 100% kompatybilne z komputerkiem edukacyjnym AVT-2250, z którym wielu z Was, drodzy

Czytelnicy pracuje już od kilku miesięcy. Zapoznamy się z kilkoma rozwiązaniami praktycznymi, a analiza przykładowych procedur obsługi tych układów pozwoli na zrozumienie zasady ich działania oraz nauczyć każdego elastycznie dołączać dodatkowe urządzenia wejścia-wyjścia do dowolnego układu mikroprocesorowego, opartego o procesory serii '51, a także podobne.



Co to są właściwie te układy wejścia/wyjścia i do czego służą? Najprostszą odpowiedzią na to pytanie niech będzie przykład, kiedy to za pomocą mikroprocesora (pracującego np. w układzie komputerka edukacyjnego - z zewnętrzną magistralą danych) chcemyysterować wiele (np. 20) żarówek, zapalając je na przemian, tak jak to się dzieje w przypadku popularnych węży świetlnych. Każdy w tym miejscu powie, że do tego potrzebne będą układy pośredniczące (tzw. mocy), bowiem jak wiemy obciążalność prądowa wyjść portów procesora jest niewielka, zresztą żarówki pracują w większym zakresie napięć, toteż dołączenie ich bezpośrednio do mikrokontrolera z pewnością przyczyniło by się do jego uszkodzenia. Jest to prawda, ale czy do końca? Odpowiedź z pewnością znajdziecie w niniejszym artykule.

Rys. 1 Dołączenie sekcji żarówek do procesora 8051



## PROSTE UKŁADY WYJŚCIOWE

Na rys.1 pokazano przykładowe dołączenie 8-miu niskonapięciowych żarówek do jednego, wolnego portu procesora 8051, pracującego z zewnętrzną pamięcią programu - tak jak nasz komputer edukacyjny. Jak widać w naszym przykładzie zastosowano, jako bufor mocy popularny układ ULN2803, który jest ośmiokrotnym buforem mocy, dodatkowo zabezpieczony diodami zwrotnymi przed przepięciami.

Zamiast takiego układu, można np. posłużyć się bramkami TTL z otwartym kolektorem i wyjściami wysokonapięciowymi. Do takich układów należą np.:

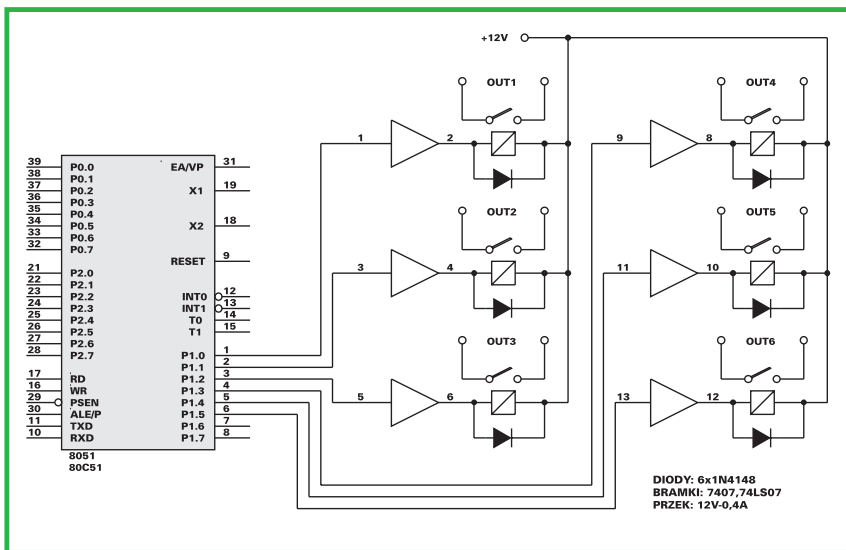
7406 - 6-krotny inwerter z wysokonapięciowym wyjściem (do +30V) typu otwarty kolektor. Ociążalność każdej bramki wynosi dodatkowo 10 wejść TTL.

7407 - 6-krotny wzmacniacz, bufor, z wysokonapięciowym wyjściem (do +30V) typu otwarty kolektor. Obciążalność każdej bramki wynosi 25 wejść TTL.

7438 - 4 dwuwejściowe braki NAND typu otwarty kolektor ze zwiększoną obciążalnością do 30 wejść TTL.

Nie będę tu przytaczał schematów wewnętrznych tych układów, wszystkie one mniej lub bardziej dokładnie zostały omówione w cyklu artykułów "Pierwsze kroki w cyfrowce" w EdV. Na rys.2 przedstawiono sposób na zwiększenie obciążalności wyjść procesora za pomocą jednego z w/w układów do sterowania sekcją 6-ciu przekaźników. Do tego celu zaprzężnięto 1 układ 7407. Dodatkowo każdą cewkę przekaźnika zbocznikowano diodą małosygnałową w celu tłumienia przepięć tworzących się w cewce podczas zdejmowania napięcia z jej zacisków.

## Też to potrafisz



Rys.2 Sterowanie przełącznikami za pomocą portu P1 procesora.

Aby zilustrować sterowanie taką sekcją przełączników posłużmy się przykładem. Otóż aby załączyć np. tylko wyjścia OUT1, OUT2 i OUT5, należy wykonać instrukcję:

```
MOV P1, #101100b ;wyzerowanie bitów - końcówek 0, 1 i 4 portu P1
```

Wyzerowanie odpowiednich końcówek portu P1 spowoduje "otwarcie" wyjścia odpowiedniej bramki i załączenie przełącznika. Te przełączniki, które mają pozostać wyłączone, powinny mieć ustawione odpowiednie bity rejestru portu P1. W naszym przykładzie są to bity 2, 3 i 5 (bity numeruje się od zera), co odpowiada przełącznikom OUT3, OUT4 i OUT6.

Można także włączyć lub wyłączyć dowolny przełącznik oddzielnie, jaką instrukcją, to zapewne już wiecie, a no chociażby włączenie przełącznika OUT4 realizowane jest za pomocą polecenia:

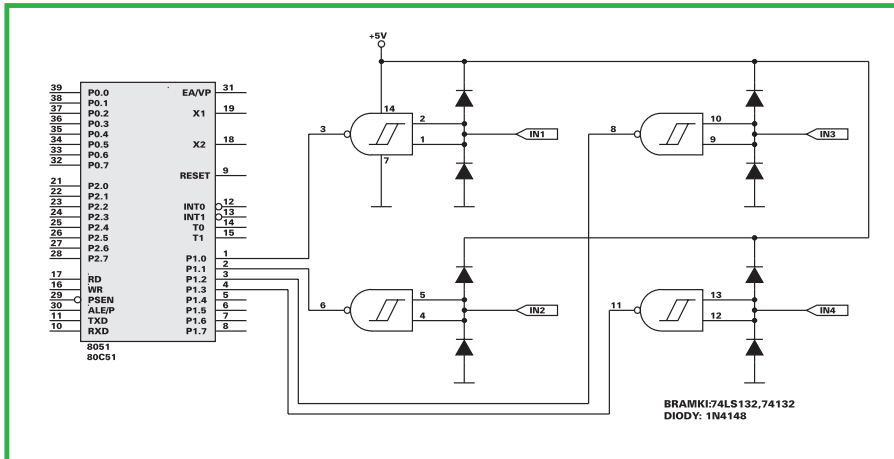
```
CLR P1.3 ;włączenie przełącznika OUT4
```

a wyłączenie

```
SETB P1.3 ;wyłączenie przełącznika OUT4
```

Zauważmy że takie sterowanie przełącznikami "od plusa" zasilania (kiedy cewka znajduje się po stronie dodatniego napięcia a nie masy) w układach procesorowych dodatkowo zabezpiecza przełączniki, przed przypadkowym, krótkotrwałym włączeniem w przypadku, kiedy uruchamiamy cały układ. Jak bowiem wiecie z poprzednich odcinków klasy mikroprocesorowej, procesor po włączeniu zasilania "zeruje się" i na wszystkich jego portach wstępnie ustawiają się stany wysokiej

Rys.3 Najprostszy sposób na bezpieczny monitorowanie sygnałów niskonapięciowych z zakresu -0,5...+5,5V.



impedancji, co dla bramek TTL jest po prostu jedynką. Dlatego w takiej sytuacji przełączniki pozostaną wyłączone.

Dlatego w tak prostych układach sterowania, np. przełącznikami nie zaleca się stosowania bramek np. 7406. Wspominałem o tym już w pierwszych dwóch numerach klasy mikroprocesorowej ponad rok temu.

Przedstawione powyżej przypadki to najprostsze z możliwych przykłady urządzeń wyjścia (ang. "Output") w układach mikroprocesorowych. I nie są to bynajmniej wspomniane żarówki, czy przełączniki oraz towarzyszące im elementy, ale "to co nimi steruje". W tym przypadku jest to port P1 procesora - to on był właśnie układem wyjścia, wyposażone dodatkowo w bufor zwiększające jego obciążalność.

## PROSTE UKŁADY WEJŚCIOWE

Często zachodzi potrzeba monitorowania stanów logicznych na kilku liniach jednocześnie i podejmowania w związku z tym określonych działań. Najprostszym przykładem jest "obserwacja" przez procesor linii dozorowych w systemie alarmowym. Na

rys.3 pokazano uzbrojone wejścia portu P1, jako najprostszy układ wejściowy procesora. Rolę buforów wejściowych pełnią bramki zabezpieczone dodatkowo na wejściach przez przebiegi diodami małosygnałowymi lub transilami. Te ostatnie to nowoczesne elementy potrafiące "gasić" przepięcia sięgające kilkuset woltów i trwające mikrosekundy - a więc bardzo krótko, wystarczająco jednak długo aby zniszczyć bramkę, a co za tym idzie cały układ scalony. Oczywiście nie muszę w tym miejscu przestrzegać przez bezpośrednim dołączeniem wysokonapięciowych linii zewnętrznych do wejść procesora, może mieć to bowiem złe skutki i spowodować awarię całego systemu.

W układzie w celu wyeliminowania stanów przejściowych zastosowano bramki Schmitta zawarte w popularnym układzie 74132. Taki system pozwala na monitorowanie sygnałów cyfrowych nadchodzących np. ze znacznej (jak na układy cyfrowe) odległości, czyli już od 50 cm w górę nawet do kilku metrów. Układy wejściowe dostosowane do większych napięć przedstawimy przy innej okazji, już teraz można jednak stosować chociażby np. proste dzielniki napięcia dodatkowo transilami lub diodami.

Odczyt stanu na danym wejściu możliwy jest podobnie jak w przypadku wyjść. Można więc odczytać cały port na raz np. za pomocą instrukcji:

```
MOV A, P1
```

a potem analizować poszczególne bity akumulatora, lub monitorować każdy pin portu oddzielnie. Nie należy jednak zapomnieć, aby przedtem zainicjować końcówkę portu jako wejście wpisując do rejestru tego portu logiczną jedynkę, za pomocą instrukcji:

```
SETB P1.x
```

gdzie x - oznacza numer końcówki portu P1. Sytuacja taka jest konieczna tylko wtedy, kiedy od momentu uruchomienia (zasilenia) procesora, port P1 (lub niektóre jego piny) były wykorzystywane jako wyjścia pracujące w stanie logicznego "0".

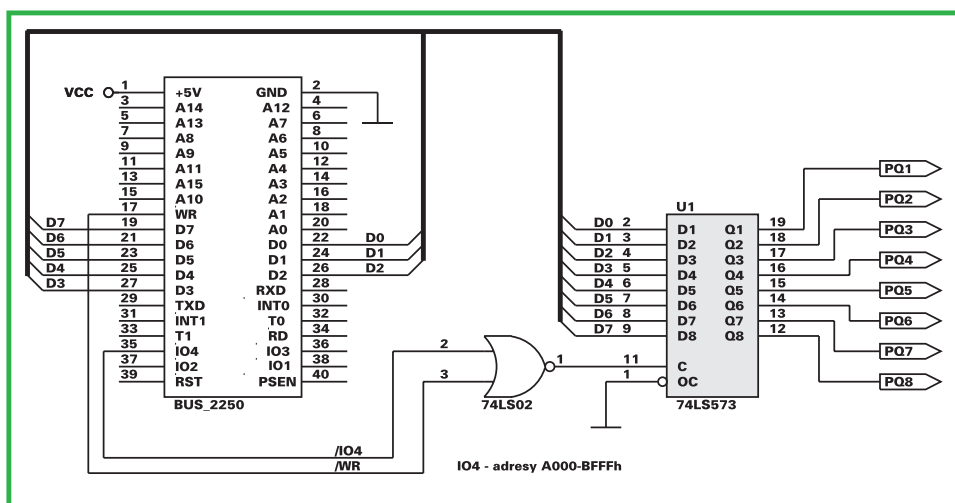
Toteż jeżeli przedtem użyłeś polecenia np.

```
CLR P1.5 ;ustawienie stanu logicznego "0" na końcówce 6 procesora
```

zerowałeś jakieś piny portu P1 poleceniem

```
MOV P1, xx
```





Rys.4 Dołączenie zewnętrznego portu wyjściowego

podanie wysokiego poziomu logicznego na wejście C (pin 11) układu 74LS573 powoduje przeniesienie stanów logicznych panujących na wejściach D1...D8 tego układu odpowiednio na wyjścia Q1...Q8. Wejście OC steruje trójstanowymi wyjściami Q1...Q8. Podanie stanu niskiego na tę końcówkę powoduje odblokowanie wyjść Q1...Q8, dzięki czemu zatrzaśnięte z wejść D1...D8 stany logiczne pojawią się na wyjściach układu U1.

Utrzymywanie tej końcówki w stanie logicznej "1" powoduje ustawienie wszystkich wyjść w stan wysokiej impedancji.

W tabeli 1 przypominam zasadę działania układu 74573. Znaczenie symboli jest następujące:

Z – stan wysokiej impedancji  
H – stan wysoki (logiczny)  
L – stan niski (logiczny)  
x – stan nieistotny (dowolny)

gdzie xx to albo stała albo rejestr zawierający liczbę z wyzerowanymi bitami

powinno, przed użyciem tego pinu jako wejścia, ustawić w rejestrze portu P1 jego bit, np. poleceniem:

```
SETB P1.5
```

tylko raz, a potem odczytywać stan końcówki, gdy zajdzie potrzeba przy pomocy instrukcji np.

```
MOV C, P1.5
```

lub testować końcówkę portu wprost za pomocą instrukcji np. skoków warunkowych

```
JB P1.5, ustawiony
```

..... ; instrukcje, gdy stan na pinie 5 portu P1 jest niski

ustawiony:

..... ; instrukcje, gdy stan na tej końcówce jest wysoki

## ... ZA MAŁO..., ODWIECZNY PROBLEM

W praktyce, kiedy mamy do czynienia z procesorami 8051 pracującymi z zewnętrzną pamięcią programu, do wykorzystania pozostaje jedynie port P1 oraz niektóre końcówki portu P3. Porty P0 i P2 są zajęte generowaniem sygnałów na szynie adresowej w celu odczytu kolejnych instrukcji z pamięci programu - EPROM, lub odczytem / zapisem danych do zewnętrznej pamięci danych (SRAM). Wtedy w najlepszej sytuacji mamy do wykorzystania wolne piny:

- P1.0... P1.7 - 8 końcówek

- P3.2 (INT0), P3.3 (INT1), P3.4 (T0), P3.5 (T1), P3.0 (RXD), P3.1 (TXD) - 4 końcówki

- P3.6 (WR) i P3.7 (RD) - dodatkowo 2 końcówki, jeżeli nie korzystamy z zewnętrznej pamięci danych

W sumie więc mamy do dyspozycji 16 uniwersalnych końcówek wejścia/wyjścia (I/O). Niestety często okazuje się że to stanowczo za mało. Wtedy z pomocą może nam przyjść rozbudowanie układów wejścia/wyjścia za pomocą dodatkowych układów cyfrowych (np. serii TTL), dołączonych do magistrali procesora.

Dzięki takim układom oraz za pomocą instrukcji odwołujących się do zewnętrznej pamięci danych (MOVX.....) możliwe jest "ustawianie" lub odczytywanie większej - praktycznie dowolnej ilości końcówek cyfrowych.

Na rys.4 przedstawiony jest dobudowany do magistrali komputerka edukacyjnego prosty układ wyjściowy. BUS\_2250 to złącze - magistrala komputerka znajdująca się na płytce bazowej w postaci 2-rzędowego 40-pinowego złącza, na jej krawędzi.

Zastosowano układ scalony, 74LS573, który jest 8-bitowym latch'em, znanym z konstrukcji komputerka AVT-2250, gdzie pełnił rolę zatrzaśki młodszej części 16-bitowego adresu. Dla przypomnienia powiem, że

Tabela 1

| wejścia |   |   | wyjścia   |
|---------|---|---|-----------|
| OC      | C | D | Q         |
| H       | x | x | Z         |
| L       | L | x | bez zmian |
| L       | H | L | L         |
| L       | H | H | H         |

Ponieważ w naszym przykładzie układ U1 pracuje jako wyjściowy, końcówkę OC dołączyliśmy do masy, tak aby na wyjściach panował stan - zatrzaśnięty z wejść D1...D8 podczas opadającego zbocza sygnału na wejściu "C" (pin 11).

Nie wchodząc na początku w szczegóły powiem, że układ z rys.4 pozwala na zapamiętanie stanów cyfrowych na ośmiu wyjściach (Q1...Q8) układu U1 za pomocą instrukcji:

```
MOV DPTR, #IO4
```

```
MOV A, #10101010b
```

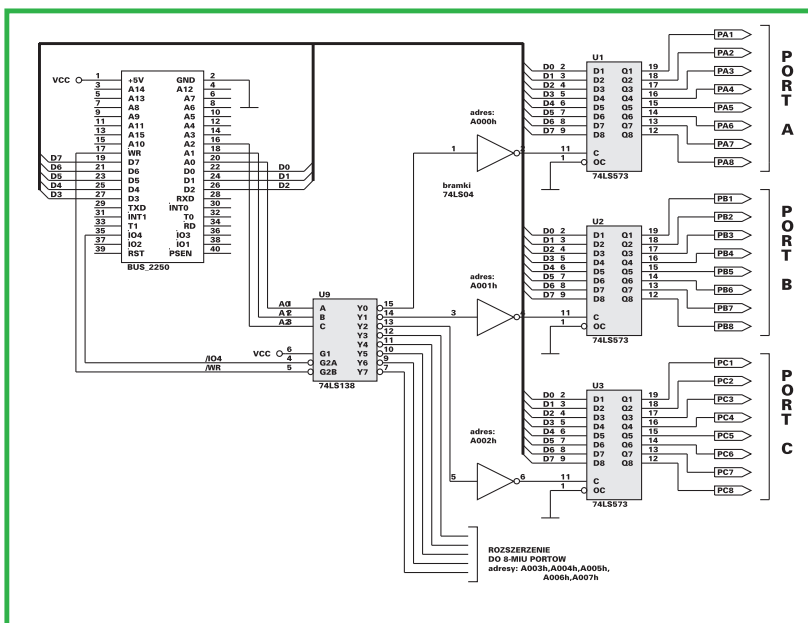
```
MOVX @DPTR, A
```

W przykładzie tym wykonanie instrukcji spowoduje ustawienie na przemian stanów logicznych wysokich i niskich na końcówkach Q1...Q8 portu U1. Stanie się tak za sprawą zapisania liczby 10101010b pod adres w zewnętrznej przestrzeni adresowej od adresu IO4 (A000h).

Przeanalizujmy zapis do układu U1. Procesor wykonując instrukcję MOVX,... powoduje pojawienie się stanu niskiego na końcówce IO4 złącza komputerka w przypadku kiedy w rejestrze DPTR znajdzie się dowolny adres z zakresu A000h...BFFFh. Równocześnie w momencie zapisu do rejestru procesor podaje stan niski na linię /WR, co w efekcie powoduje powstanie stanu wysokiego na wyjściu bramy 74LS02 (rys.4), która realizuje negację sumy logicznej sygnałów /WR i /IO4. Wysoki stan na wyjściu tej bramki powoduje zapisanie danych D0...D7 z szyny danych procesora 8051 do rejestru U1, a dzięki temu, że wejście /OC U1 jest zwarte do masy, dane te pojawią się natychmiast na wyjściach Q1...Q8 zatrzaśki U1.

W ten sposób za pomocą jednej bramki logicznej, jednego układu scalonego (U1) oraz trzech instrukcji zrealizowaliśmy prosty układ portu wyjściowego o ośmiu wyjściach. Wyjścia te, podobnie jak opisywanego wcześniej portu P1, można teraz wykorzystać do dowolnych celów, np. do sterowania układami wykonawczymi dowolnych urządzeń, np. 8-kanalowego węża świetlnego.

## Też to potrafisz



Rys.5 Jeden ze sposobów na rozbudowanie ilości układów I/O procesora.

Zapisując dowolne, zależne od sytuacji wartości za pomocą instrukcji podanych w ostatnim przykładzie (w miejsce #10101010b) możemy wpływać na stany 8-miu wyjść portu U1, a dzięki temu sterować maksymalnie 8-ma urządzeniami zewnętrznymi.

Na rys.5 przedstawilem sposób na powiększenie ilości układów wyjściowych. Jak widać do tego celu wykorzystano dodatkowy układ dekodera 74LS138 (U9). Dzięki takiemu rozwiązaniu, za pomocą 1 dekodera '138 możliwe jest zaadresowanie maksymalnie 8-miu rejestrów typu 74LS573. Aby zrozumieć schemat ideowy najlepiej jest znaleźć podobieństwa z rys.4. W tym przypadku dekodery U9 dekodują 3 najmłodsze linie adresowe magistrali procesora: A2, A1 i A0. Dzięki temu każdy z rejestrów 74LS573 może być zapisany oddzielnie, za pomocą sekwencji instrukcji podanych w poprzednim przykładzie, z tym że inne będą wartości wpisywane do rejestru DPTR. I tak aby np. zaadresować (zapisać) rejestr U3 należy posłużyć się instrukcjami:

```
MOV DPTR, #IO4_U3
```

```
MOV A, #11001100b
```

```
MOVX @DPTR, A
```

wcześniej należy jednak umieścić deklarację opisującą zapis: IO4\_U3, tzn.

```
IO4_U3 equ A002h
;bo IO4=A000h +2 (offset U3)
```

Można także zapisać adres od razu do rejestru DPTR, instrukcją:

```
MOV DPTR, #A002h
```

a efekt będzie ten sam.

W układzie z rys.5 sygnały IO4 i /WR zostały dołączone do dwóch wejść zezwalających dekodera U9, dzięki czemu w momencie gdy jeden i drugi przyjmuje stan '0' dekodery zostają odblokowane i uaktywnione zostaje (stanem niskim) jedno z 8-miu wyjść układu U9, zależnie od stanu linii adresowych A2...A0.

Stan niski z wyjścia dekodera U9 zostaje zaneigowany przez dołączoną do niego bramkę negacji (NOT) a następnie w postaci dodatkowego impulsu (zapisu) powoduje zatrzęsienie danych D0...D7 z magistrali komputerka w jednym z rejestrów 74LS573. Reasumując aby "wysterować" poszczególne rejestry, i każde z ich 8-miu wyjść, należy do DPTR wpisać ich odpowiednie adresy, i tak:

dla U1 - MOV DPTR, #0A000h

dla U2 - MOV DPTR, #0A001h

dla U3 - MOV DPTR, #0A002h

dla U8 - MOV DPTR, #0A007h

a następnie wykonać zapis do akumulatora stanu 8-miu wyjść danego rejestru, np.:

```
MOV A, 1111110b ;tylko wyjście Q8 w stanie "0", reszta "1"
```

i wykonać zapis

```
MOVX @DPTR, A
```

i to wszystko, prawda że proste!

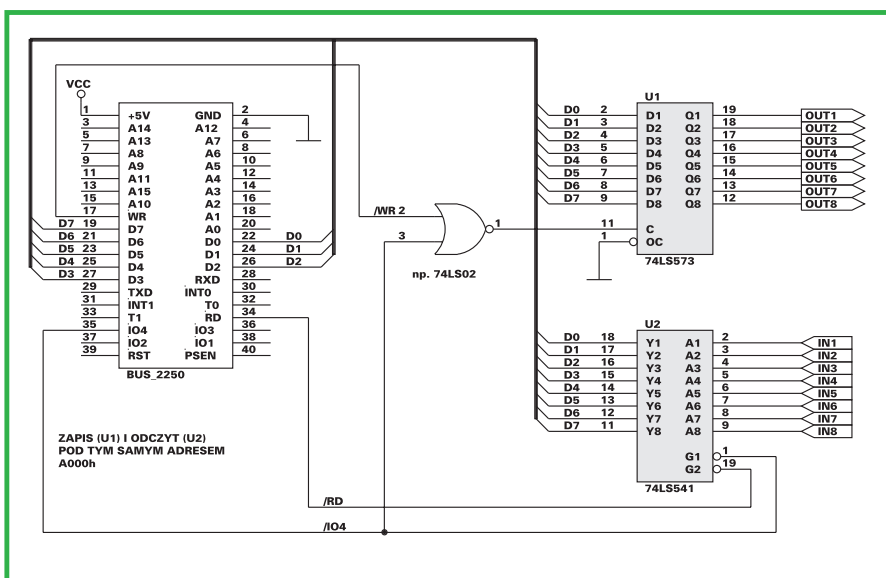
Na rys.5 nie pokazano, ze względu na oszczędność miejsca dołączenia pozostałych 5-ciu rejestrów U4...U8, ale sposób sterowania i podłączenia do dekodera jest identyczny jak w przypadku U1...U3, czyli do wyjść dekodera U9 poprzez bramki negacji.

We wszystkich pokazanych przykładach można używać układów w wersjach CMOS, np. 74HCT138, 74HCT04, 74HCT573, itp., należy jednak pamiętać aby uwzględnić mniejszą obciążalność wyjść tych ostatnich przy dołączaniu dodatkowych układów wyjściowych. Uwaga ta będzie dotyczyć także pozostałych przedstawionych w artykule przykładów.

No dobrze, układy wyjściowe, umożliwiające sterowanie zewnętrznymi urządzeniami mamy w zasadzie omówione, no przynajmniej te najprostsze rozwiązania. Pora przejść do układów, które umożliwią odczyt zewnętrznych stanów logicznych przez procesor, krótko mówiąc "badanie" cyfrowych sygnałów dochodzących z zewnątrz do układu komputerka.

Rysunek 6 przedstawia prosty sposób na sterowanie zapisem i odczytem 8-miu zewnętrznych linii cyfrowych. Spójrzmy przez chwilę jedynie na zastosowanych tu nowy układ scalony U2 – 74LS541.

Jest to 8-mio bitowa "brama", dzięki której możliwy jest transfer danych z jej



Rys.6 Sposób na wykorzystanie tego samego adresu do sterowania zapisem (U1) i odczytem (U2) sygnałów cyfrowych.

wejść (A1...A8) na wyjścia (Y1...Y8) w momencie, kiedy oba sygnały sterujące G1, G2 znajdują się w stanie niskim. Innymi słowy, kiedy podamy na wejścia G1, G2 stan niski, to "to" co pojawi się na wejściu (np. A1) pojawi się także na wyjściu Y1. W przeciwnym przypadku, gdy jedno (lub oba) wejścia znajdują się w stanie wysokim, wszystkie wyjścia Y1...Y8, znajdując się z stanie wysokiej impedancji. Dzięki temu m.in. możliwe jest zastosowanie tych układów w systemach mikroprocesorowych, takich jak komputerek edukacyjny.

Tabela 2 opisuje działanie układu.

Tabela 2

| wejścia |    |   | wyjścia |
|---------|----|---|---------|
| G1      | G2 | A | Y       |
| H       | x  | x | Z       |
| x       | H  | x | Z       |
| L       | L  | L | L       |
| L       | L  | H | H       |

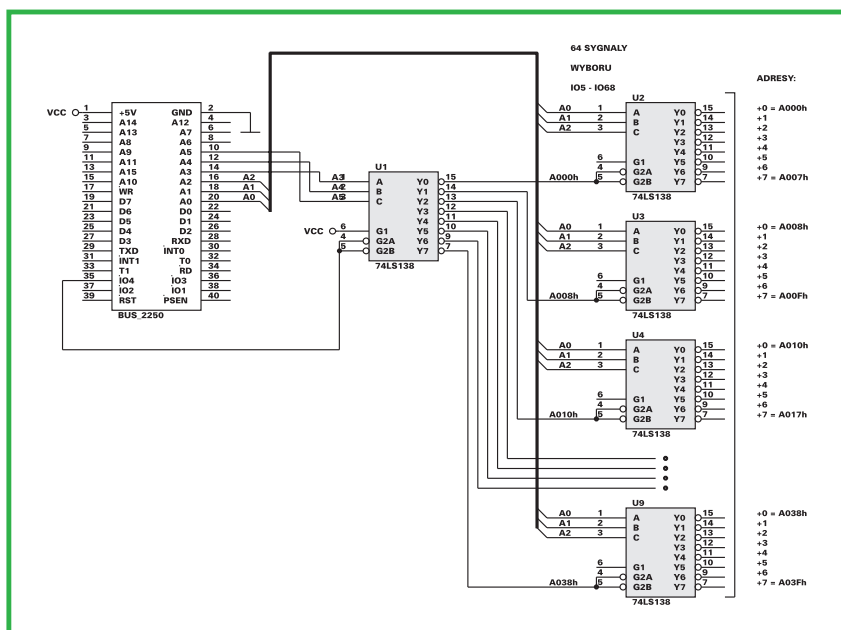
Popatrzmy na tę część schematu, która odnosi się do układu U2, bowiem sposób podłączenia U1 i bramki NOR został już omówiony. Podobnie jak w przypadku układu wyjściowego (U1) podłączony został bufor wejściowy U2. W tym przypadku obyło się jednak bez korzystania z dodatkowej bramki sumy logicznej, bowiem, wykorzystano dwa wejścia zezwalające G1 i G2 układu 74LS541. Jedno z tych wejść dołączono do sygnału /RD procesora, a drugie do sygnału /IO4, który jak wiemy przyjmuje stan niski w momencie adresowania obszaru A000h...BFFFh procesora. W ten prosty sposób, kiedy np. chcemy odczytać stany wejść A1...A8 układu U2, możemy to zrobić za pomocą instrukcji:

```
MOV DPTR, #IO4
```

```
MOVX A, @DPTR
```

i to wszystko. W akumulatorze znajdzie się liczba, której poszczególne bity odpowiadają stanom na wejściach A1...A8 w momencie odczytu (tzn. kiedy sygnał /RD przyjmuje stan niski) przez mikroprocesor portu U2.

Rys.8 Sposób na kaskadowe łączenie dekodów 74138



Rys.7 Zastosowanie układu demultipleksera jako dekodera adresów

W tym miejscu bardziej wnikliwi czytelnicy mogą dostrzec problem, który może pojawić się jeżeli w obszarze adresowym dekodowanym przez sygnał IO4 pojawi się inny układ, np. pamięć (lub jej część) statyczna SRAM. Wtedy odczytując port U2 (adres 0A000h) odczytana może zostać komórka w pamięci SRAM. W przypadku gdy obie dane pochodzące z portu i pamięci nie będą zgodne nastąpi efekt iloczynny montażowego, który w najbardziej niekorzystnym przypadku (przy użyciu układów serii LS) może spowodować uszkodzenie pamięci lub portu U2. Dlatego w przykładach omawianych w niniejszym artykule, należy sprawdzić, aby pamięć SRAM w komputerze miała pojemność 8kB (typ 6264). W przypadku gdy posiadasz pamięć 62256 (32kB), należy przełączyć zworę JP3 na płytce bazowej komputerka na adres C000h a programy testujące porty I/O opisane w artykule, kompilować oczywiście z dyrektywą:

```
ORG C000h
```

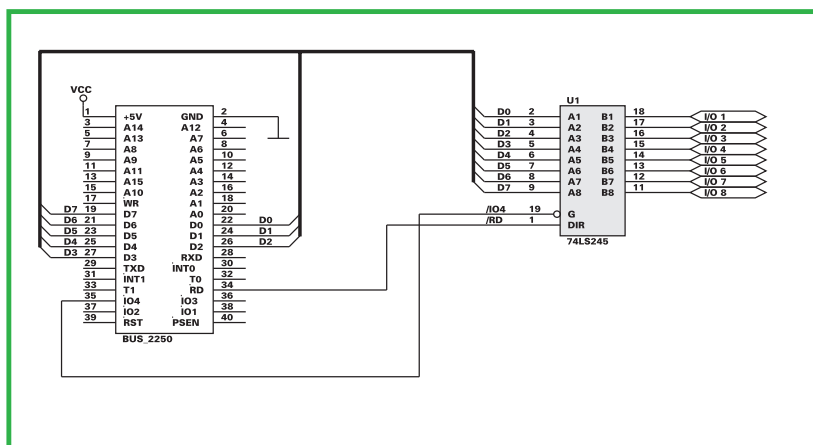
Na rys.7 przedstawiono sposób na zwiększenie liczby sygnałów wyboru I/O do 16-tu za pomocą zwykłego układu demultipleksera 74LS154. Dzięki takiemu dołączeniu linii adresowych oraz sygnału /IO4 komputerka do układu U1, w prosty sposób uzyskujemy dodatkowe linie sterujące dowolnymi urządzeniami np. I/O, takimi jakimi opisałem wcześniej. W układzie z rys.7 poszczególne wyjścia IO5...IO20 odpowiadają następującym adresom:

IO5, adres A000h  
IO6, adres A001h  
IO7, adres A002h  
IO8, adres A003h  
IO9, adres A004h  
IO10, adres A005h  
IO11, adres A006h  
IO12, adres A007h  
IO13, adres A008h  
IO14, adres A009h  
IO15, adres A00Ah  
IO16, adres A00Bh  
IO17, adres A00Ch  
IO18, adres A00Dh  
IO19, adres A00Eh  
IO20, adres A00Fh

Na kolejnym rysunku 8 pokazany jest sposób na generowanie wielu sygnałów wyboru I/O za pomocą kaskadowo połączonych dekodów 74138. Jak widać, dzięki układowi U1 możliwe jest wysterowanie kolejnych dekodów U2...U9 i w efekcie uzyskanie 64 sygnałów wyboru. Analizując powyższy schemat należy zwrócić uwagę na sposób dołączania poszczególnych linii adresowych do wejść A, B i C dekodów 74138.

Zrozumienie zasady łączenia wejść adresowych pozwoli na samodzielne, dowolne organizowanie przestrzeni adresowej procesora. Kolejne przykłady przybliżą Ci to, drogi Czytelniku.

## Też to potrafisz



Rys.9 Zastosowanie dwukierunkowej bramy logicznej

Rysunek 9 przedstawia zastosowanie trzeciego ciekawego układu scalonego z popularnej serii TTL, mianowicie dwukierunkowej 8-mio bitowej bramy logicznej.

Podobnie jak poprzednio wyjaśniam, że układ 74LS245 potrafi przekazywać dane z wejść A1...A8 na wyjścia B1...B8 i odwrotnie, tzn. z B1...B8 (które wtedy są wejściami) na A1...A8 (które pełnią rolę wyjść), w zależności od poziomu na końcówce DIR (pin 1). Tabela 3 przedstawia zależności logiczne układu.

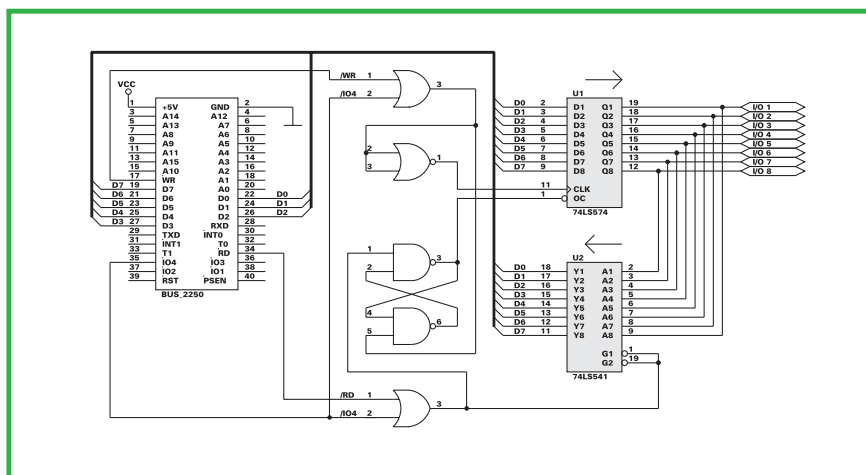
Tabela 3

| wejścia |     | funkcja   |
|---------|-----|-----------|
| G       | DIR |           |
| H       | H   | A = B = Z |
| H       | L   | A = B = Z |
| L       | H   | A -> B    |
| L       | L   | A <- B    |

Jak widać z tabeli podanie stanu wysokiego na wejście DIR układu powoduje przekazanie danych z A1...A8 odpowiednio na wyjścia B1...B8, i odwrotnie podanie stanu niskiego powoduje przekazanie stanów z B1...B8 na A1...A8. Sygnał G służy do uaktywniania układu. Podanie na ten pin stanu wysokiego powoduje ustawienie końcówek A1...A8 i B1...B8 w stan wysokiej impedancji.

Należy zwrócić uwagę, że układ 74LS245 nie jest zatraskiem, tzn. że nie zapamiętuje danych z wejść, lecz je tylko "przekazuje" na wyjścia.

Rys.10 Dwukierunkowy port I/O – okrojona wersja portu mikroprocesora



W układzie przykładowym z rys.9 sterowanie kierunkiem przepływu informacji następuje pod wpływem sygnału /RD procesora, dołączonego do wejścia DIR układu bramy U1. W takiej konfiguracji układ wykorzystywany jest działa podobnie jak 74LS541, przekazuje dane z wejść B1...B8 na wyjścia A1...A8 a dalej na szynę danych procesora (komputerka) D0...D7. Kiedy procesor adresuje bufor poprzez sygnał zapisu (/WR) uaktywniony zostaje sygnał IO4, który powoduje przekazanie stanów z szyny danych komputerka D0...D7 na końcówki B1...B8 bramy U1. Ktoś może zauważyć, że taki sposób obsługi układu w trybie zapisu może być "nieelegancki", bowiem w procesie aktywacji układu U1 nie jest wykorzystywany sygnał zapisu /WR. I choć na pierwszy rzut oka jest to prawdą, to w praktyce takie sterowanie wystarcza. Ja jednak proponuję Ci, drogi Czytelniku, jako ćwiczenie narysowanie pełnego dekodera dla przykładu z rys.9.

Na rys.10 przedstawiłem ciekawy układ dwukierunkowego portu I/O z możliwością zapamiętania wartości w trybie zapisu. Działanie portu jest identyczne z funkcjami portów mikroprocesora (np. omawianego wcześniej P1), jednak w tym przypadku możliwe jest używanie wszystkich 8-miu bitów portu jednocześnie jako wyjścia lub wejścia. Nie ma więc możliwości ustawiania pojedynczych końcówek portu (jak to ma miejsce w przypadku procesora) jako wejście a innych jako wyjścia w tym samym momencie. Jednak często taka funkcja nie jest nam potrzebna, toteż powyższy układ często znajduje swoje zastosowanie w praktyce i przyznam spisuje się w niej znakomicie.

Można zatem porównać ten układ do portu procesora, np. P1 i powiedzieć, że instrukcja zapisu typu:

```
MOV DPTR, #IO4
```

```
MOV A, #11110000b
```

```
MOVX @DPTR, A
```

działa podobnie jak instrukcja

```
MOV P1, #11110000b
```

oczywiście w odniesieniu do dwóch różnych układów wejścia-wyjścia.

Podobnie instrukcje:

```
MOV DPTR, #IO4
```

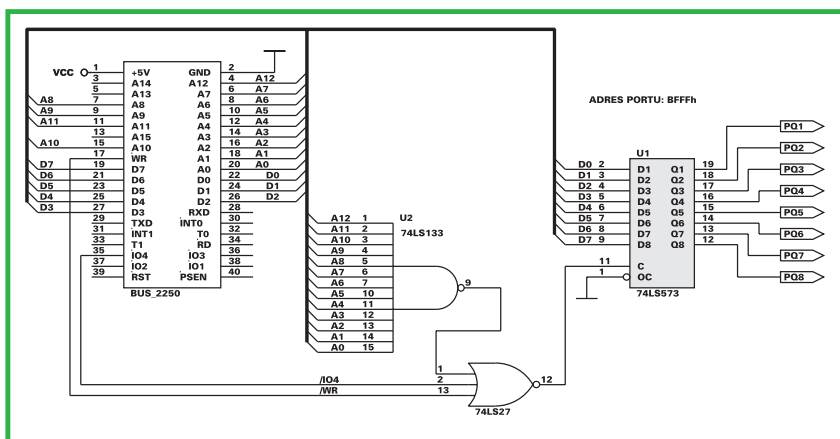
```
MOVX A, @DPTR
```

powodują odczytanie stanów końcówek portu z rys.10, co dla portu procesora można było wykonać instrukcją:

```
MOV A, P1
```

Zamiast znanego już układu 74LS573 zastosowano podobnie działający układ 74LS574. Różnica w działaniu obu układów polega na tym, że pierwszy układ "przepuszcza dane" przez cały czas, kiedy na wejście C podany jest stan wysoki, a zapamiętuje je podczas opadającego zbocza sygnału na tym wejściu. Drugi układ jest typowym zatraskiem wyzwalanym zboczem, stąd w symbolu graficznym zamiast oznaczenia C na końcówce 1 znajduje się symbol zegara CLK. Tak więc dane są zapamiętywane w układzie w momencie narastającego zbocza sygnału na tym wejściu. Działanie układu ilustruje także tabela 4. W praktyce w większości zastosowań możliwe jest zastosowanie obu układów zamiennie – zgodna jest reszta topologia pozostałych wyprowadzeń obu kostek.





Rys.11 Przykład pełnego dekodowania pojedynczego układu I/O

Tabela 4

| wejścia |      |   | wyjścia   |
|---------|------|---|-----------|
| OC      | CLK  | D | Q         |
| H       | x    | x | Z         |
| L       | L    | x | bez zmian |
| L       | L->H | L | L         |
| L       | L->H | H | H         |

Popatrzmy na rys.10. W stanie spoczynku, sygnały /RD, /WR i /IO4 są w stanie wysokim. Dzięki temu na wejściu zapisu układu U1 (pin 11) panuje stan niski, dzięki bramce NOR negującej sumę logiczną sygnałów /WR i /IO4 (która dokonywana jest za pomocą bramki OR).

Dodatkowo wejście kasujące przerzutnika /RS zbudowanego z 2 bramek NAND jest w stanie wysokim. Podobnie sygnał /RD jest w stanie wysokim a po zsumowaniu z sygnałem /IO4 ustawia w stanie wysokim wejście zezwalające G1, G2 układu U2, czyli blokuje działanie bramy U2. Jeżeli teraz procesor wykona cykl zapisu do portu, to w wyniku tego nastąpi zapis do układu U1 oraz dodatkowo zostanie ustawiony przerzutnik /RS, dzięki czemu na wejściu zezwalającym U1 (OC) powstanie stan niski, co uaktywni wyjścia tego układu i w efekcie pojawienie się danych z szyny danych komputerka D0...D7 na wyjściach I/O 1...I/O 8 całego układu. Cykl zapisu nie wpłynie przy tym na działanie układu wejściowego U2, który pozostanie nadal nieaktywny. Jeżeli procesor dokona cyklu odczytu, to stan niski z wyjścia bramki OR sumującej sygnały /RD i /IO4 spowoduje skasowanie przerzutnika /RS i pojawienie się stanu wysokiego na wejściu zezwolenia /OC U1 – co zablokuje układ U1 (wyjścia w stanie wysokiej impedancji). Dzięki temu możliwy będzie odczyt stanów linii

I/O 1...I/O 8. Ze względu na pewne niedoskonałości przedstawionego układu z rys.10, polegające na nieuwzględnieniu niektórych zależności czasowych, konieczne okazać się może dwukrotne odczytanie portu IO4 w momencie, gdy ostatnio był on używany jako wyjście. Dzieje się tak dlatego, że może nastąpić sytuacja, kiedy przerzutnik /RS nie zdąży przerzucić swego stanu i zablokować wyjścia układu U1, zanim procesor odczyta stan linii wyjściowych portu. W takim przypadku odczytana zostanie ostatnia wartość wpisana do portu U1.

Dlatego jeżeli ostatnią operacją na układzie z rys.10 było np.

```
MOV DPTR, #IO4
```

```
MOVX @DPTR, A ; zapis do portu U1
```

to przy pierwszy odczyt warto przeznaczyć "na straty"

```
MOVX A, @DPTR
```

a dopiero drugi

```
MOVX A, @DPTR
```

potraktować jako prawidłowy i dokonać po tym analizy akumulatora.

## ACH, TE ADRESOWANIE...

Z pewnością niektórzy z Was, drodzy Czytelnicy, zauważyli, analizując powyższe przykłady, fakt tzw. "niepełnego dekodowania adresów". Chodzi o to, że adresując – zapisując układ, np. z rys.4, mamy do

czynienia z pojedynczym układem wyjściowym, czyli fizycznie zajmującym tylko jeden adres w przestrzeni adresowej procesora, a tymczasem, sposób podłączenia go do komputerka pozwala na użycie aż 8192 adresów (A000h...BFFFh). Dzieje się tak dlatego, bo wykorzystany do selekcji układu U1 (rys.4) sygnał /IO4 jest aktywny dla tego właśnie zakresu adresów. Toteż zaadresowanie

```
MOV DPTR, #A000h
```

```
MOVX, @DPTR, A
```

czy

```
MOV DPTR, #BFFFh
```

```
MOVX, @DPTR, A
```

czy może

```
MOV DPTR, #B260h
```

```
MOVX @DPTR, A
```

w praktyce da ten sam efekt.

Można więc z całą odpowiedzialnością stwierdzić że jest to czyste "marnotrawstwo" adresów! I tak z pewnością jest. Aby temu zaradzić stosuje się pełniejsze lub całkiem pełne dekodowanie danego adresu (lub kilku) tak aby np. port U1 z rys.4 był dekodowany tylko i wyłącznie dla danego jednego adresu np. BFFFh – i żadnego więcej. W ten sposób w pozostałym obszarze A000h...BFFFh można by umieścić pozostałe 8191 układów I/O, co nie zawsze w praktyce jednak jest potrzebne. Na rys.11 pokazano zmodyfikowany układ dekodowania pojedynczego portu wyjściowego (z rys.4). Jak widać układ U1 zostanie zmodyfikowany tylko jeżeli procesor zaadresuje go pod adresem: BFFFh. Dzieje się tak dlatego, że dodatkowa bramka U2 realizuje negację iloczynu logicznego portu wyjściowego (z rys.4). Jak widać układ U1 zostanie zmodyfikowany tylko jeżeli procesor zaadresuje go pod adresem: BFFFh. Dzieje się tak dlatego, że dodatkowa bramka U2 realizuje negację iloczynu logicznego portu wyjściowego (z rys.4). Jak widać układ U1 zostanie zmodyfikowany tylko jeżeli procesor zaadresuje go pod adresem: BFFFh. Dzieje się tak dlatego, że dodatkowa bramka U2 realizuje negację iloczynu logicznego portu wyjściowego (z rys.4).

```
MOVX @DPTR, A
```

kiedy to także sygnał /WR przyjmie stan niski. Wtedy na wyjściu bramki U2 pojawi się niski stan logiczny, który wraz z niskim stanem sygnału /IO4 oraz niskim stanem /WR spowoduje pojawienie się wysokiego poziomu na wyjściu trzywejściowej bramki NOR.

W przykładzie zastosowano nietypową, ale spotykaną w handlu 13-wejściową bramkę logiczną NAND typu 74LS133. W razie trudności z nabyciem, można w zastępstwie użyć połączonych kaskadowo wielu bramek NAND o mniejszej liczbie wejść. Można także spróbować zawęzić obszar dekodowania układu I/O w inny sposób, możliwości jest wiele, toteż inne rozwiązania pozostawiam wam jako pracę domową, drodzy Czytelnicy.

W dzisiejszym odcinku to tyle na temat prostych układów wejścia-wyjścia. Następnym razem zajmiemy się bardziej złożonymi – programowalnymi układami I/O, dzięki którym możliwy jest nie tylko odczyt i zapis ale także generowanie przerwań. Pokażę wam także jak w prosty sposób wyposażać nasz komputer edukacyjny w port drukarkowy zgodny ze standardem Centronics – w dodatku pracujący w dwóch kierunkach!

Przedstawię schemat ideowy takiego rozwiązania oraz procedury w języku 8051 realizujące pożyteczne funkcje drukowania wprost z komputerka edukacyjnego oraz dodatkowo emulujące złącze drukarki w komputerku, dzięki czemu możliwy będzie transfer danych np. z komputerem PC do komputerka AVT-2250 tym razem nie przez złącze RS-232c ale przez złącze równoległe Centronics i to z wielą większą prędkością! Na razie to tyle, zapraszam więc do kilku dodatkowych zadań, które przygotowałam na powakacyjne jesienne już wieczory.

Ślawomir Surowiński

## LEKCJA 10

Proponuję trzy zadania. Oto one:

1. Zmodyfikować układ z rys. 9 tak aby uwzględnił przy sterowaniu sygnał /WR zapisu, bez zmiany funkcjonalnej układu – patrz tekst.
2. Narysować zmodyfikowany układ dekodera z rys.7 który zamiast układu 74LS154 wykorzystuje układy 74LS138.
3. Do układu z rys. 10 dobudować dekodery, dzięki któremu układ U1 będzie zapisywany pod adresem A000h a odczytywany spod adresu A001h.

Rozwiązania zadań w kolejnym numerze EdW. Wesolej zabawy!



# Mikrokontrolery?

## To takie proste...

W poprzednim odcinku zapoznaliśmy się z prostymi układami I/O, dzięki którym możliwe jest sterowanie zewnętrznymi urządzeniami za pomocą mikrokomputera edukacyjnego.

W przykładach wykorzystałem najpopularniejsze układy serii TTL (HCT-TTL), toteż zastosowanie ich w praktycznych układach nie powinno nastręczać problemów.

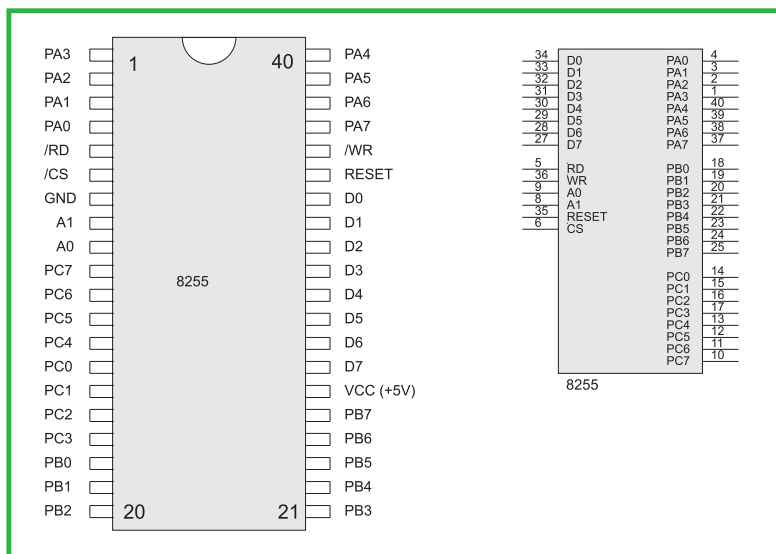
Tym razem mam zamiar przedstawić bardziej rozbudowane układy, umożliwiające np. drukowanie na typowej drukarce komputerowej, bądź wykorzystanie szybkiej transmisji równoległej z urządzenia zewnętrznego do komputera edukacyjnego.

### Część 18

#### Programowane układy wejścia-wyjścia

Układ który mam zamiar dzisiaj przedstawić jest tak stary jak cała technika komputerowa, oczywiście ta z prawdziwego zdarzenia, czyli z przełomu lat 70-tych i 80-tych. Mowa będzie o scalaku oznaczanym przez producentów jako 8255 – programowany układ wejścia/wyjścia z łączem równoległym. Zanim jednak przejdę do części praktycznej i przedstawię konkretne schematy oraz listingi procedur obsługi układu 8255 wypada mi krótko przedstawić sam układ, tak aby nikt nie czuł niedosytu wiedzy w tym zakresie. Jeżeli zaś któryś z Was drodzy Czytelnicy jest na tyle niecierpliwy, że nie może się powstrzymać od skonfrontowania wiedzy praktycznej w rzeczywistości, może przejść od razu do akapitu "Drukowanie z komputera".

Rys.1 Topografia wyprowadzeń układu 8255 i jego schemat funkcjonalny



### 8255 – Dużo funkcji niewielkim kosztem

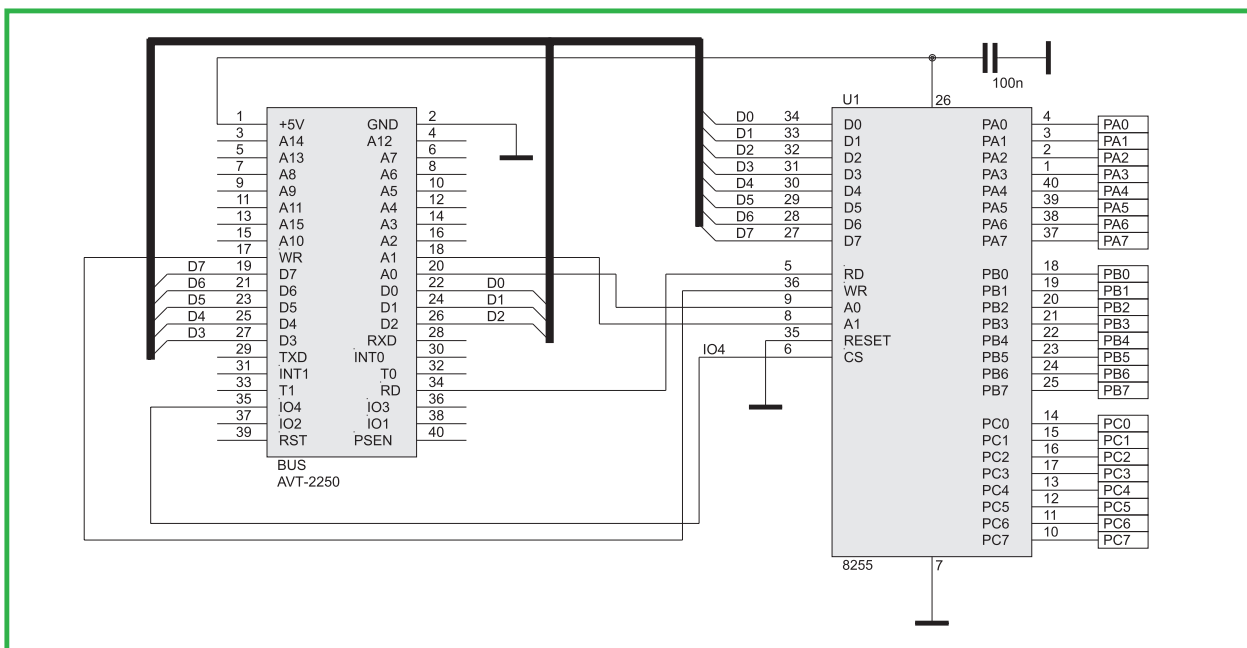
Bohater naszego dzisiejszego odcinka to jegomość umieszczony w 40-nóżkowej typowej obudowie DIL. Dzięki temu zastosowania i montaż tego układu na płytce drukowanej, podobnie jak procesorów 8051/2 nie nastręcza trudności. Ma to szczególnie znaczenie bo w dzisiejszej erze miniaturyzacji, na rynku rozpanoszyły się wszelkiego rodzaju zminiaturyzowane obudowy, które niejednego amatora elektroniki mogą przyprawić o zawrót głowy.

W czasach, kiedy technika komputerowa PC wkraczała pod strzechy, kiedy to IBM wyprodukował pierwszy komputer osobisty, a w chwilę później pojawiło się na rynku wielu innych producentów sprzętu, w układach elektronicznych komputerów PC stosowano układy 8255 bardzo często. Pierwszym zastosowaniem była obsługa łącza równoległego standardu Centronics, przeznaczonego pierwotnie dla potrzeb drukowania. W chwili obecnej kiedy wszystkie układy I/O komputera PC znalazły się na "płyce głównej", układ 8255 wraz z całą rodziną wcześniej produkowanych scalaków z mikroprocesorowej rodziny Intel'a znalazł swoje miejsce z raczej został "wcielony" do tzw. układów "chip set'u" płyty komputera.

Pomimo tego, że w dzisiejszych komputerach PC do sterowania portem równoległym kostki 8255 nie są już wykorzystywane, to jednak można je spotkać w przemysłowych wersjach komputerów sterujących - także PC. Stary, ale sprawdzony standard przyjął się tak dobrze jak poczcziwa już pięćdziesiątka-jedynka i na razie nic nie zapowiada jej upadku. Układ 8255 jest nadal produkowany i można go nabyć prawie w każdym sklepie ze specjalistycznymi artykułami elektronicznymi. Koszt kostki mieszczący się na poziomie około 5..10 zł nie jest kwotą wygórowaną, a pamiętając że układ za tak niską cenę oferuje wiele możliwości, dostajemy go praktycznie za darmo.

Najprościej rzecz mówiąc, układ 8255 zawiera trzy uniwersalne 8-bitowe porty (PA, PB i PC), z których każdy może być skonfigurowany jako wejście, wyjście a w pewnych trybach spełniać rolę mieszana, nawet z możliwością generowania przerwań do procesora. Dodatkowo, oprócz wspomnianych

## Też to potrafisz



Rys.2 Sposób dołączenia kostki 8255 do komputerka edukacyjnego.

rejestrów portów PA, PB i PC układ 8255 zawiera tzw. czwarty rejestr – konfiguracyjny, dzięki któremu możliwe jest ustalenie, jak ma pracować cały układ, czy np. port PA ma być ustawiony jako wejściowy, czy wyjściowy itp.

Dołączenie układu do komputerka edukacyjnego (generalnie do procesorów 8051/2) jest bardzo proste, bowiem 8255 sterowany jest sygnałami kompatybilnymi z tymi generowanymi przez mikrokontrolery serii MCS-51. Sposób dołączenia kostki do komputerka edukacyjnego omówię za chwilę. Tymczasem na rys.1 przedstawiam rozkład wyprowadzeń kostki 8255.

Jak widać do połączenia układu 8255 z urządzeniami zewnętrznymi służą linie PA0...PA7, PB0...PB7 i PC0...PC7. To w jaki sposób ustawione są te linie oraz ich funkcje, determinuje specjalne słowo (bajt) konfiguracyjne (wspomniany czwarty rejestr) - zaszyte wewnątrz układu 8255. Jak to słowo zapisywać, innymi słowy jak konfigurować cały układ powiem za chwilę. Na razie wyjaśnijmy sobie znaczenie poszczególnych wyprowadzeń układu. Co prawda jest ich aż 40-ci ale jak się za chwilę przekonasz, drogi Czytelniku, nie jest to wcale dużo.

**D0...D7** – linie dołączane do szyny danych systemu mikroprocesorowego. Dzięki nim, możliwa jest transmisja danych z procesora nadzorującego do i z układu 8255. W przypadku komputerka edukacyjnego AVT-2250 linie te powinny dołączyć się do szyny adresowej D0...D7 komputerka – wyprowadzenia: 22,24,26,27,25,23,21,19 złącza BUS AVT-2250

**/CS** – sygnał wejściowy wyboru – selekcji układu przez procesor. Podanie logicznego zera na to wejście umożliwia transmisję danych poprzez linie D0...D7. W układzie komputerka edukacyjnego sygnał ten może być dołączony do jednego z wyjść dekodera adresowego IO3 lub IO4, tak jak w przykładach z poprzedniego odcinka klasy mikroprocesorowej.

**/WR** – sygnał zapisu danych do układu 8255 przez procesor zewnętrzny. Podanie stanu niskiego na te wejście powoduje zapisanie danych z szyny D0...D7 do wewnętrznych rejestrów układu 8255. W układzie komputerka edukacyjnego sygnał ten łączy się bezpośrednio z końcówką /WR procesora (lub pinem 17 złącza BUS na płycie głównej AVT-2250).

**/RD** – sygnał odczytu danych z układu 8255. Podanie stanu niskiego na te wejście spowoduje odczytanie informacji z wewnętrznych rejestrów układu 8255. Podobnie jak w przypadku sygnału zapisu, wejście to powinno się dołączyć do wyjścia /RD procesora 8051 (lub do pinu 34 złącza BUS komputerka AVT-2250).

**A1, A0** – linie sterujące wyborem jednego z czterech rejestrów wewnętrznych układu 8255. W zależności od poziomów na tych liniach podczas cyklu odczytu/zapisu przez procesor wybrany zostaje jeden z trzech rejestrów wyjściowych PA, PB, PC bądź rejestr konfiguracyjny.

**RESET** – wejście zerowania układu 8255. Podanie stanu wysokiego na to wejście spowoduje wyzerowanie rejestrów wewnętrznych

(rejestr sterującego) układu 8255 i ustawienie portów PA, PB, PC jako wejściowych. Taki sam stan układu 8255 ustawiany jest automatycznie po każdorazowym włączeniu zasilania.

**PA0...PA7** – linie 8-bitowego uniwersalnego portu PA (pierwszego portu)

**PB0...PB7** – j/w lecz portu PB (drugiego portu)

**PC0...PC7** – j/w lecz portu PC (trzeciego portu)

Na rys.2 przedstawiony jest najprostsz sposób dołączenia układu 8255 do komputerka AVT-2250. Jeżeli śledzisz uważnie cykl klasy mikroprocesorowej, w szczególności jeśli zapoznałeś się z poprzednim odcinkiem o prostych układach wejścia/wyjścia, z pewnością stwierdzisz, że sposób połączenia jest oczywisty i analogiczny z układami z poprzedniego odcinka klasy.

I tak linie danych D0...D7 dołączamy do szyny danych komputerka BUS – D0...D7. Linie sterujące zapisem /WR i odczytem /RD do odpowiednich linii złącza BUS (WR i /RD). Sygnał wyboru układu /CS dołączamy bezpośrednio do wyjścia dekodera adresowego w układzie komputerka np. IO4, natomiast linie wyboru rejestru do analogicznych linii adresowych szyny adresowej komputerka A0 i A1. Prawda że proste! I to już wszystko, moi drodzy, można zająć się programowaniem układu. Aha byłby zapomnian o konieczności dołączenia zasilania, które przyłącza się trochę nietypowo (uwaga dla tych którzy w tym momencie jedną ręką i jednym okiem chcą natychmiast zabrać się za wykonanie płytki drukowanej), mianowicie +5V dołącza się do końcówki 26 (a nie jak typowo w obudowach DIL do pinu 40) a masę do końcówki 7.

Projektując płytkę lub umieszczając układ na "uniwersalce" warto zablokować linię zasilającą tuż przy układzie 8255 kondensatorem 100nF.

W ten sposób po dołączeniu układu do komputerka mamy do dyspozycji trzy 8-mio bitowe porty PA, PB i PC, do których możemy dołączyć dowolne urządzenia.

Wnikliwy czytelnik z pewnością zauważy konsekwencję z zastosowania dwóch linii adresowych A0 i A1 w układzie 8255 ze względu na sterowanie 4-roma rejestrów wewnętrznych układu 8255. Tabela 1

Tabela 1

| A1 | A0 | wybrany rejestr 8255                           |
|----|----|------------------------------------------------|
| 0  | 0  | rejestr portu PA                               |
| 0  | 1  | rejestr portu PB                               |
| 1  | 0  | rejestr portu PC                               |
| 1  | 1  | rejestr kontrolny (CTRL) – konfiguracji układu |



przedstawia znaczenie sygnałów A0 i A1 w odniesieniu do wyboru jednego z czterech rejestrów wewnętrznych kostki 8255.

W połączeniu ze sterowaniem sygnałem wyboru /CS poprzez linię np. IO4 poszczególne rejestry 8255 będą widziane przez procesor komputerka jako adresy:

PA – adres A000h (A1,A0 = 0,0)

PB – adres A001h (A1,A0 = 0,1)

PC – adres A002h (A1,A0 = 1,0)

CTRL – adres A003h (A1,A0 = 1,1)

Analogicznie, jeśli podłączylibyśmy linię wyboru /CS np. do sygnału IO3 komputerka to wtedy jak się zapewne domyślasz, adresy wyglądałyby odpowiednio tak: 8000h, 8001h, 8002h, 8003h.

Toteż, jeżeli po włączeniu zasilania komputerka z dołączonym do niego układem 8255, procesor (51-ka) wykona instrukcję:

MOV DPTR, #A000h ; adres rejestru portu PA

MOVX A, @DPTR ; odczyt rejestru PA z 8255

to w efekcie w akumulatorze znajdzie się liczba której poszczególne bity będą zgodne z poziomami sygnałów na liniach PA0...PA7 układu 8255. Podobnie można będzie odczytać linie PB0...PB7 w następujący sposób:

MOV DPTR, #A001h

MOV A, @DPTR

i rejestru C

MOV DPTR, #A002h

MOV A, @DPTR

O ile rejestr PA (a także PB) może pełnić rolę wejść lub wyjść cyfrowych (nie jest możliwe aby niektóre bity tych rejestrów były wejściami a inne wejściami jednocześnie – w tej samej chwili), o tyle rejestr PC ma szczególne dodatkowe właściwości. Otóż może on być tak skonfigurowany (za pomocą rejestru kontrolnego CTRL), że w jednocześnie z portu PC można odczytywać dane lub je zapisywać. Oczywiście w takim przypadku wybrane linie PC będą pracowały wyłącznie jako wejścia, pozostałe zaś jako wyjścia. Fizycznie w układzie 8255 w odróżnieniu do rejestrów PA i PB, które widziane są jako całe 8-mio bitowe rejestry, rejestr PC może być modyfikowany w dwóch "połówkach". I tak starsze cztery bity rejestru PC7...PC4 będą oznaczane jako PCa młodsze PC3...PC0 jako PCb. Obrazowo wszystkie 3 rejestry portów przedstawiam poniżej.

## Rejestr PA - końcówki 1-4 i 37-40 układu 8255

|          |     |     |     |     |     |     |     |     |           |
|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----------|
| nr bitu: | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |           |
| bity     | PA7 | PA6 | PA5 | PA4 | PA3 | PA2 | PA1 | PA0 | <b>PA</b> |

## Rejestr PB - końcówki 18-25 układu 8255

|          |     |     |     |     |     |     |     |     |           |
|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----------|
| nr bitu: | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |           |
| bity     | PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 | <b>PB</b> |

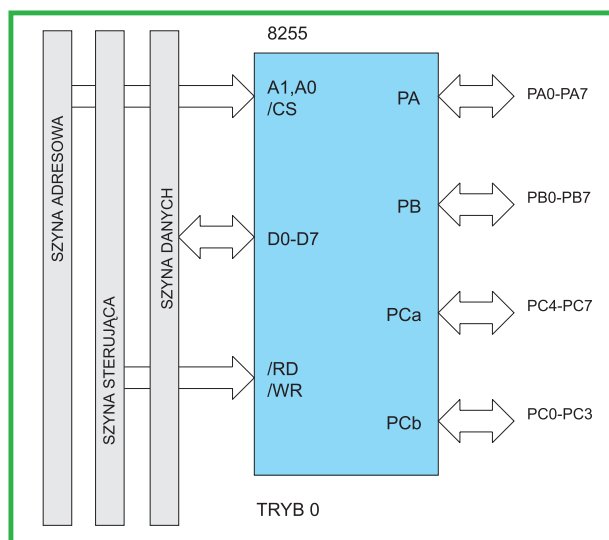
## Rejestr PC - końcówki 10-17 układu 8255

|          |             |     |     |     |             |     |     |     |           |
|----------|-------------|-----|-----|-----|-------------|-----|-----|-----|-----------|
| nr bitu: | 7           | 6   | 5   | 4   | 3           | 2   | 1   | 0   |           |
| bity     | PC7         | PC6 | PC5 | PC4 | PC3         | PC2 | PC1 | PC0 | <b>PC</b> |
|          | rejestr PCa |     |     |     | rejestr PCb |     |     |     |           |

Ważną informacją jest to że odczyt rejestru kontrolnego CTRL (pod adresem A003h) jest, uwaga, zabroniony! Rejestr ten można tylko zapisywać. A czym go zapisywać? Otóż zapoznamy się teraz z budową tego rejestru i jego bitami oraz ich znaczeniem dla pracy kostki 8255.

## Rejestr kontrolny 8255 (CTRL) w trybie konfigurowania

|          |   |    |    |    |    |    |    |    |             |
|----------|---|----|----|----|----|----|----|----|-------------|
| nr bitu: | 7 | 6  | 5  | 4  | 3  | 2  | 1  | 0  |             |
| bity     | 1 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | <b>CTRL</b> |



Rys.3 Układ 8255 w trybie pracy 0

W trybie programowania układu najstarszy bit (7) rejestru powinien być zawsze ustawiony ("1"). Znaczenie pozostałych bitów jest następujące:

bit 7 – w trybie programowania układu 8255 zawsze "1".  
bity D6 i D5 – określają wybór trybu pracy układu i rejestrów PA oraz PCa, i tak:

00 = tryb 0

01 = tryb 1

10 lub 11 = tryb 2. Znaczenie poszczególnych trybów omówię za chwilę.

bit D4 – określa kierunek pracy linii portu PA i tak

1 = PA jako wejście

0 = PA jako wyjście cyfrowe

W trybie 2 pracy układu 8255, bit ten nie ma znaczenia.

bit D3 – określa kierunek pracy linii portu PCa (PC7...PC4) i tak

1 = PCa jako wejście

0 = PCa jako wyjście cyfrowe

bit D2 – określa wybór trybu pracy rejestru PB, i tak:

0 = tryb 0

1 = tryb 1

bit D1 – określa kierunek pracy linii portu PB, i tak:

1 = PB jako wejście

0 = PB jako wyjście cyfrowe

bit D0 – określa kierunek pracy linii portu PCb (linie PC3...PC0) i tak

1 = PCb jako wejście

0 = PCb jako wyjście

Jak widać regułą jest, że ustawienie bitów D0, D1, D3, D4 powoduje ustawienie odpowiadających im portów jako wejścia, natomiast wyzerowanie tych bitów powoduje ustawienie tych portów jako wyjścia cyfrowe. W najprostszym trybie pracy mamy zatem aż 16 kombinacji ustawień

portów PA, PB i dwóch połówek portu PC.

## TRYB 0

Wspomniane tryby pracy (0, 1 lub 2) umożliwiają wybór sposobu działania całego układu. Dla uproszczenia zajmijmy się na razie najprostszym trybem 0. W trybie tym jak powiedziałem wcześniej każdy z rejestrów PA, PB oraz dwóch połówek PC (PCa i PCb) może pracować jako wejście lub wyjście cyfrowe. Dane przesyłane są z szyny danych D0...D7 systemu mikroprocesorowego do rejestrów portu (ustaw-

## Też to potrafisz

| bajt sterujący<br>CTRL (binarnie) | zapis<br>hex | port<br>PA | port<br>PB | port PC         |                 |
|-----------------------------------|--------------|------------|------------|-----------------|-----------------|
|                                   |              |            |            | linie PC7...PC4 | linie PC3...PC0 |
| 100 1 1 0 1 1                     | 9Dh          | WE         | WE         | WE              | WE              |
| 100 0 1 0 1 1                     | 8Dh          | WY         | WE         | WE              | WE              |
| 100 1 1 0 0 1                     | 99h          | WE         | WY         | WE              | WE              |
| 100 0 1 0 0 1                     | 89h          | WY         | WY         | WE              | WE              |
| 100 1 0 0 1 1                     | 93h          | WE         | WE         | WY              | WE              |
| 100 0 0 0 1 1                     | 83h          | WY         | WE         | WY              | WE              |
| 100 1 0 0 0 1                     | 91h          | WE         | WY         | WY              | WE              |
| 100 0 0 0 0 1                     | 81h          | WY         | WY         | WY              | WE              |
| 100 1 1 0 1 0                     | 9Ah          | WE         | WE         | WE              | WY              |
| 100 0 1 0 1 0                     | 8Ah          | WY         | WE         | WE              | WY              |
| 100 1 1 0 0 0                     | 98h          | WE         | WY         | WE              | WY              |
| 100 0 1 0 0 0                     | 88h          | WY         | WY         | WE              | WY              |
| 100 1 0 0 1 0                     | 92h          | WE         | WE         | WY              | WY              |
| 100 0 0 0 1 0                     | 82h          | WY         | WE         | WY              | WY              |
| 100 1 0 0 0 0                     | 90h          | WE         | WY         | WY              | WY              |
| 100 0 0 0 0 0                     | 80h          | WY         | WY         | WY              | WY              |

**Tabela 2**

ionych jako wyjścia) lub odczytywane poprzez tą szynę do procesora w sposób niesynchronizowany.

Rys.3 obrazuje ideę pracy układu 8255 w trybie 0.

W przypadku kiedy procesor ustawi np. rejestr PA jako wyjście cyfrowe, to zapisując potem ten rejestr za pomocą instrukcji:

```
MOVX @DPTR,A
```

powoduje że dane z szyny danych D0...D7 pojawią się na liniach portu PA7...PA0, zgodnie z daną umieszczoną w akumulatorze Acc). W przypadku zaś odwrotnym, kiedy np. rejestr PA pracuje jako wejściowy, odczyt rejestru za pomocą instrukcji

```
MOVX A,@DPTR
```

spowoduje pojawienie się na liniach D0...D7 szyny danych (a potem w akumulatorze Acc) stanów linii portu PA7...PA0).

I tak dla przykładu zapiszmy słowo sterujące CTRL układu 8255 w taki sposób żeby np.

port PA i PC pracowały jako WYJŚCIA  
port PB jako WEJŚCIE,

patrząc na opis bitów rejestru (wyżej) zapiszemy:

D6, D5 i D2 będą = 0 (tryb 0)  
D4 = 0 to PA będzie wyjściem  
D3 = 0 to PCa – wyjście  
D1 = 1 to PB – wejście

wreszcie

D0 = 0 to PCb jako wyjście (podobnie jak PCa)

zatem aby tak skonfigurować układ 8255 należy wykonać sekwencję instrukcji:

```
MOV A, #10000010b ;bity D7, D1 = 1, reszta "0"
MOV DPTR, #A003h ;zakładam, że /CS jest dopięty do /IO4 komputera
MOVX @DPTR, A ;i zapisanie konfiguracji do układu 8255
```

**Rejestr kontrolny 8255 (CTRL) w trybie modyfikacji rejestru PC**

| nr bitu: | 7 | 6  | 5  | 4  | 3  | 2  | 1  | 0  |      |
|----------|---|----|----|----|----|----|----|----|------|
| bity     | 1 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | CTRL |

Aby ułatwić konfigurowanie układu w trybie pracy 0 w praktycznych zastosowaniach, poniżej zamieszczam tabelę 2, w której zapisane są wszystkie wartości rejestru kontrolnego CTRL, które dają jeden z 16 sposobów ustawień portów PA, PB i PC.

W tabeli dla zwiększenia czytelności zapisałem wartość wpisywaną do rejestru kontrolnego CTRL w postaci binarnej oraz szesnastkowej. Dodatkowo rozdzieliłem spacjami niektóre bity w zapisie binarnym, tak abyś mógł łatwiej przeanalizować sposób ustawienia lub zerowania poszczególnych bitów zgodnie z wcześniejszym opisem.

Teraz jeżeli chcemy ustawić porty PA, PB i PC jako wyjścia (np. do sterowania 24-oma diodami LED) wystarczy sięgnąć do tabeli i odczytać liczbę 80h jako stosowną do zapisu do rejestru konfiguracyjnego układu 8255. Potem wystarczy wykonać ciąg instrukcji:

```
MOV A, #80h
MOV DPTR, #A003h
MOVX @DPTR, A ;i układ 8255 skonfigurowany
```

Teraz jeżeli chcemy np. "zapalić" linie portu PA, to wystarczy wydać polecenie

```
MOV A, #255
MOV DPTR, #A000h
MOVX @DPTR, A
```

W praktyce przy pisaniu programu źródłowego (komputerowcy) warto posłużyć się deklaracją EQU dla zdefiniowania poszczególnych portów układu 8255. Wtedy nie trzeba będzie za każdym razem pamiętać, co jakie adresy mają porty PA, PB czy PC. W naszym przykładzie można to zrobić w sposób następujący:

```
IO_PA EQU A000h ;deklaracja adresu portu PA
IO_PB EQU A001h ;i/w lecz portu PB
IO_PC EQU A002h ;i/w lecz portu PC
IO_CTRL EQU A003h ;wreszcie rejestru kontrolnego 8255
```

Teraz wystarczy odpowiednio i elegancko ich używać, np. tak:

```
MOV A, #91h ;ustawiam: PA i PCb jako WE, PB i PCa jako WY
MOV DPTR, #IO_CTRL ;adres rejestru kontrolnego
MOVX @DPTR, A ;i skonfigurowanie układu
```

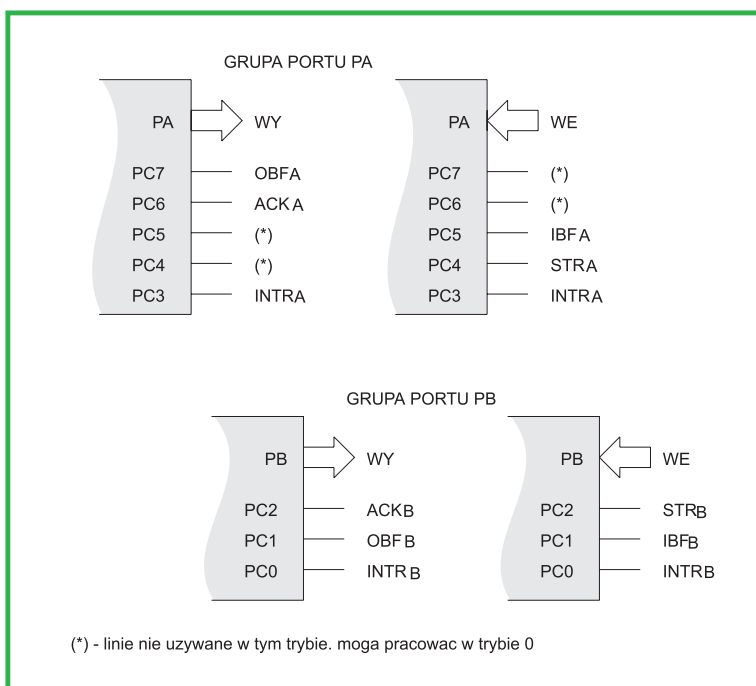
```
MOV DPTR, #IO_PA ;adres rejestru PA
MOVX A, @DPTR ;i odczyt linii PA7...PA0
```

```
MOV DPTR, #IO_PB ;adres rejestru PB
MOV A, #55h ;ustaw naprzemiennie linie PB7...PB0 jako "01010101"
MOVX @DPTR, A ;i zapis do rejestru PB
..... ;itd.
```

Na szczególną uwagę zasługuje rejestr PC układu 8255. Otóż możliwe jest selektywne ustawianie bądź zerowanie poszczególnych bitów tego rejestru. Do tego celu służy zapis do rejestru kontrolnego słowa z wyzerowanym najstarszym bitem D7. Poniżej przedstawiam strukturę słowa rejestru CTRL dla takiego przypadku sterowania.

W trybie tym najstarszy bit (7) rejestru powinien być wyzerowany. Znaczenie pozostałych bitów jest następujące:

bit 7 – zawsze "0".  
bity D6, D5, D4 – nie używane (dowolna wartość, zazwyczaj "0")



Rys.4 Znaczenie linii portów PA, PB i PC w trybie pracy 1.

bitów D3, D2, D1 – określają numer ustawianego lub zerowanego bitu rejestru PC, i tak:

- kolejność "000" oznacza wybór bitu D0 (linia PC0)
- kolejność "001" oznacza wybór bitu D1 (linia PC1)
- kolejność "010" oznacza wybór bitu D2 (linia PC2)
- kolejność "011" oznacza wybór bitu D3 (linia PC3)
- kolejność "100" oznacza wybór bitu D4 (linia PC4)
- kolejność "101" oznacza wybór bitu D5 (linia PC5)
- kolejność "110" oznacza wybór bitu D6 (linia PC6)
- kolejność "111" oznacza wybór bitu D7 (linia PC7)
- bit D0 – powinien zawierać "1" jeżeli chcemy wybrany bit (linię portu) rejestru PC ustawić lub "0" - jeżeli wyzerować

Przykład: zapisaliśmy wcześniej w rejestrze PC następujące stany: "10101010", teraz chcemy wyzerować także linię PC7. należy więc wydać instrukcję:

```
MOV DPTR, #IO_CTRL
MOV A, #00001110b ;zeruj linię PC7
MOX @DPTR, A ;zapis do rejestru CTRL 8255
```

i gotowe! Selektowna modyfikacja bitów rejestru C możliwa jest tylko w trybie 0 pracy rejestrów PA lub PB. Jeżeli np. rejestr PA pracuje w trybie 1 lub 2, a rejestr PB w trybie 0 to można modyfikować jedynie bity rejestru PCb (z grupy B) i odwrotnie, dla PB pracującego w trybie 1, a PA w trybie 0, modyfikowalne są tylko bity PCa.

Tryby 1 i 2 pracy układu 8255 nie są tak proste jak tryb 0. Dzięki nim jednak możliwa jest synchronizacja transmisji pomiędzy układem 8255 a urządzeniem zewnętrznym. Można powiedzieć, że w praktyce trybu 0 używa się kiedy po prostu chcemy sterować zewnętrznymi urządzeniami dołączonymi (poprzez układy pośredniczące) do linii portów PA, PC czy PB, lub także odczytywać stany tych linii w dowolnym momencie. Istotną informacją jest to że w trybie 0 informacja zapisana do każdego z rejestrów portów PA, PB, PC jest zapamiętywana do kolejnej modyfikacji wybranego rejestru. Natomiast w przypadku odczytu, rejestr danego portu jest przezroczysty dla danych przesyłanych z urządzenia zewnętrznego i nie zapamiętuje ich.

Natomiast tryby 1 i 2 używane są do bardziej złożonych zastosowań, takich właśnie jak transmisja pomiędzy inteligentnymi układami peryferyjnymi, takimi jak np. drukarka, komputer, inny układ oparty o 8255 lub podobny. Zagadnienie jest bardzo rozbudowane, dlatego poniżej skupię się do skróconego omówienia sposobu działania układu 8255 w tych trybach. Przedstawione zaś w dalszej części artykułu przykłady praktycznego zastosowania kostki 8255 w celach współpracy ze zwykłą drukarką komputerową oraz do transmisji danych z innego urządzenia

zewnętrznego, pozwolą na zilustrowanie tego zagadnienia w sposób wystarczający do naszych zastosowań w przyszłych wspólnie omawianych układach.

## TRYB 1

W tym trybie pracy dane mogą być przesyłane do i z innych urządzeń poprzez porty PA lub PB. W tym trybie transmisja jest jednokierunkowa, tzn. dane mogą być odczytywane przez cały czas do kolejnej rekonfiguracji układu 8255 lub wyłącznie zapisywane. Niektóre z linii portu PC są w tym trybie przyporządkowane rejestrów PA lub PB, a służą do przesyłania sygnałów synchronizujących transmisję, między rejestrami PA lub PB a urządzeniem zewnętrznym. Rejestr PA z odpowiednimi wybranymi liniami portu PC (PCa), które synchronizują transmisję nazywa się grupą A, natomiast rejestr PB z pozostałymi liniami portu PC (PCb) – grupą B.

Tryby pracy linii grupy A i grupy B jest programowany niezależnie. Toteż gdy jeden z rejestrów PA lub PB wraz z odpowiednimi liniami portu PC może pracować w trybie 1 to tryb pracy drugiego i linii PC należącymi wraz z nimi do grupy może pracować w trybie 1 lub 0. Rys.4 pokazuje jak układ 8255 pracuje w trybie 1.

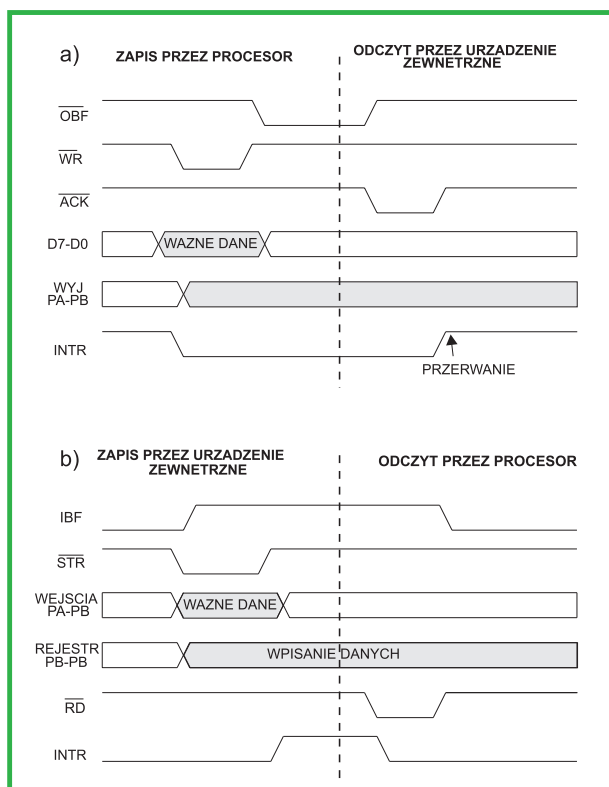
Jak wspomniałem port PC generuje na liniach dodatkowe sygnały synchronizujące transmisję danych pomiędzy 8255 a urządzeniem zewnętrznym. Znaczenie sygnałów tych jest następujące:

a) przy transmisji z układu 8255 do urządzenia zewnętrznego

**/OBF** – (ang. "output buffer full") sygnał ten przyjmuje poziom niski podczas całego cyklu przyjęcia danej z procesora (od narastającego zbocza sygnału /WR) do odebrania danej przez układ zewnętrzny (który sygnalizuje potwierdzenie odebrania znaku na linii /ACK)

**/ACK** – (ang. "acknowledge") urządzenie zewnętrzne po odebraniu znaku z portu (np. PA) potwierdza zakończenie odczytu wystawiając poziom niski na tej linii, co 8255 odbiera jako zakończenie transmisji.

Rys.5 Zależności czasowe sygnałów synchronizacji transmisji podczas odczytu danej (a) przez urządzenie zewnętrzne oraz zapisu (b) danej do układu 8255 przez to urządzenie



## Też to potrafisz

**/INTR A,B** – (ang. “interrupt”) sygnały wyjściowy 8255 zgłoszenia przerwania. Przejście tego sygnału do stanu aktywnego oznacza że urządzenie zewnętrzne odczytało przesłaną daną i jest gotowe do przyjęcia kolejnej danej. Powrót do stanu nieaktywnego następuje wtedy kiedy procesor dokona ponownie zapisu tego słowa.

b) przy transmisji danej z urządzenia zewnętrznego do 8255

**IBF** – (ang. “input buffer full”) sygnał wyjściowy który przyjmuje stan aktywny w momencie kiedy dana zostaje zapisana przez urządzenie zewnętrzne w 8255, skasowanie tego sygnału następuje kiedy procesor odczyta z 8255 tak odebraną daną.

**/STR** – (ang. “strobe”) sygnał wejściowy, dzięki któremu urządzenie zewnętrzne może zgłosić gotowość do odebrania przez 8255 danej, wystawionej przez nie

**INTR A,B** – (ang. “interrupt”) sygnały wyjściowy 8255 zgłoszenia przerwania. przejście tego sygnału do stanu aktywnego oznacza że urządzenie zewnętrzne zakończyło zapis danej w rejestrze wejściowym 8255. Powrót do stanu nieaktywnego następuje wtedy kiedy procesor dokona odczytu tej danej.

Na rys.5 a i b przedstawiono zależności czasowe odpowiednich sygnałów podczas transmisji w trybie 1. Analiza przebiegów szczególnie sygnałów sterujących nie jest trudna, należy jedynie skupić się nieco nad ideą pracy portu PC.

I tak w trybie przesyłania danej z 8255 do urządzenia zewnętrznego, procesor sterujący (u nas 8051) musi zapisać daną do układu 8255. Sygnał /WR przyjmie podczas zapisu stan niski. Równocześnie z momentem rozpoczęcie zapisu uaktywniony zostaje sygnał zgłoszenia przerwania INTR (poziom niski). Po zakończeniu zapisu danej przez procesor sygnał OBF przyjmuje stan niski, układ 8255 czeka na odebranie informacji przez urządzenie zewnętrzne. W tym czasie gotowa do odebrania dana znajduje się na wyprowadzeniach portu PA (lub PB – w zależności z której grupy A, czy B rejestrów korzystamy). Kiedy urządzenie zewnętrzne odczyta daną z portu PA (PB) zasygnalizuje ten fakt ustawieniem poziomu niskiego na linii /ACK co układ 8255 odbierze jako znak o zakończeniu odczytu daje przez te urządzenie. Sygnał /INTR zgłoszenia przerwania przyjmie stan nieaktywny (wysoki). Cykl transmisji został zakończony.

Podobnie sytuacja przedstawia się podczas transmisji znaku z urządzenia zewnętrznego do układu 8255. W tym przypadku urządzenie zewnętrzne po wystawieniu danej na port PA (PB) informuje układ 8255 o tym fakcie wystawiając poziom niski na linii /STR strobowania. W tym momencie dana ta zostaje zatrzaskana z rejestru PA (PB) a procesor dowiadyuje się o tym otrzymując np. zgłoszenie przerwania poprzez wystąpienie stanu wysokiego na wyjściu INTR 8255. Następnie procesor odczytuje daną z portu PA (PB) poprzez zaadresowania portu i podanie stanu niskiego na linii /RD. Po odczycie transmisja zostaje zakończona, sygnał IBF przyjmuje stan nieaktywny, skasowane zostaje także przerwanie INTR.

### TRYB 2

W trybie tym układ 8255 potrafi transmitować dane w dwóch kierunkach jednocześnie wraz z potwierdzeniem transmisji. Rejestr PA wykorzystywany jest wtedy do transmisji danych między 8255 a urządzeniem zewnętrznym. Jako sygnały synchronizujące wykorzysty-

wane są linie PC3...PC7. Pozostałe linie PC0...PC2 oraz rejestr PB mogą pracować niezależnie w trybie 0 lub 1.

Rys.6 ilustruje znaczenie linii portów PA i PC3...PC7 w trybie pracy 2. Znaczenie sygnałów sterujących jest takie jak w przypadku trybu 1.

### PRZERWANIA Z 8255

Jak wspomniałem wcześniej układ 8255 ma możliwość generowania przerw dla potrzeb współpracującego z nim procesora. Wiesz już że sygnały te generowane są na określonych liniach portu PC. Dodatkowo trzeba wiedzieć, że w trybie 1 każdemu z portów PA i PB przyporządkowany jest tzw. przerzutnik zezwolenia przerwania, dzięki któremu możliwe jest w ogóle wygenerowanie przerwania przez 8255. Mówiąc prościej jeżeli w swojej aplikacji masz zamiar wykorzystać sygnały INTR, to musisz odpowiednio ustawić dany przerzutnik, a robi się to bardzo prosto, bowiem poprzez zapis odpowiedniego bajtu do rejestru kontrolnego 8255. I tak dla rejestru PA jest to przerzutnik INTRA, a dla rejestru PB - INTRB.

Otóż aby ustawić dane przerzutniki i tym samym zezwolić na generowanie sygnału przerwania na linii INTR należy podobnie jak w przypadku (opisanym wcześniej) posłużyć się funkcją selektywnego ustawiania bitów rejestru PC.

W trybie 1 odpowiednie przerzutniki są przyporządkowane następującym bitom rejestru PC, i tak:

INTRB – bit D2 rejestru PC

INTRA (gdy port PA jest wejściem) – bit D4 rejestru PC

INTRA (gdy port PA jest wyjściem) – bit D6 rejestru PC

Dla przykładu podam że aby np. ustawić sygnał zezwolenia przerwania INTRB należy wykonać instrukcje:

```
MOV A, #0101b ; ustawienie przerzutnika INTRB
MOV DPTR, #IO_PC ; wybór rejestru PC
MOVX @DPTR, A ; i zapis do niego
```

oczywiście nie można zapomnieć wcześniej skonfigurować układu 8255 do pracy rejestru PB w trybie 1, można to zrobić np. poleceniem:

```
MOV A, #10011111b
MOV DPTR, #IO_CTRL
MOVX @DPTR, A
```

W przykładzie ustawiłem bit D2 akumulatora który odpowiada za tryb pracy rejestru portu PB (1- tryb 1). Pozostałe porty ustawiłem domyślnie jako pracujące jako wejścia, a PA w trybie pracy 0.

W praktyce w układach często rezygnuje się z bezpośredniego połączenia linii INTR z wejściem przerwania procesora (w 8051 jest to INTO lub INT1), ze względu na cenność wejść przerywających i możliwość monitorowania transmisji przez procesor poprzez tzw. “pulling”. Wtedy to wyjście INTR można podłączyć do jednej z linii wolnego portu 8255 pracującego w trybie 0. Cykliczny odczyt tej linii pozwoli na stwierdzenie czy transmisja została zakończona.

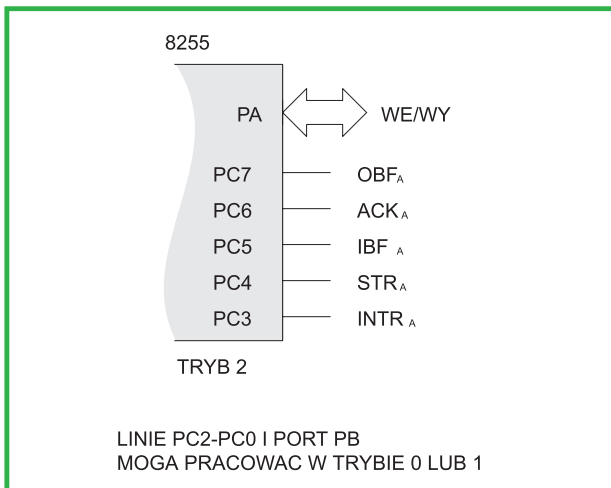
Oczywiście można stosować przerwanie procesora, lecz jak wykazuje praktyka, operacje drukowania z małych systemów automatyki nie muszą być wykonywane w tle, chociaż nic nie stoi na przeszkodzie aby tak było.

No dobrze dość już teorii, zajmijmy się praktycznymi zastosowaniami. Poza oczywistym prostym zastosowaniem układu 8255 (tryb pracy 0) przedstawie poniżej dwa ciekawe zastosowania portu, pierwsze do obsługi drukarki, drugie do emulowania drukarki przez komputer, dzięki czemu możliwe jest transmitowanie danych z komputera wyposażonego w gniazdo zgodne ze standardem Centronics wprost do AVT-2250.

### DRUKOWANIE Z KOMPUTERKA AVT-2250

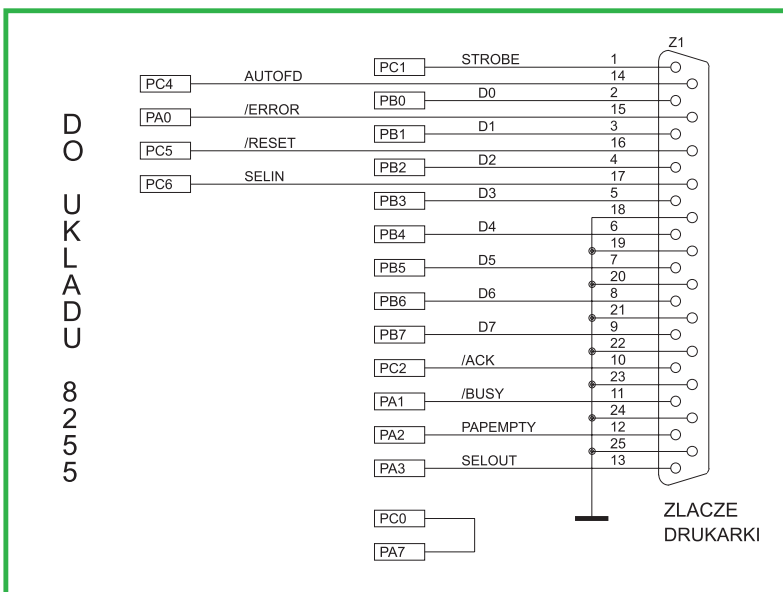
Nasz komputerka potrafi komunikować się za pomocą wyświetlacza LED oraz klawiatury lokalnej. Ciekawym uzupełnieniem, z pewnością także bardzo atrakcyjnym także dla zażradszych o twój pierwszy system mikroprocesorowy, kolegów będzie możliwość obsługi drukarki. Będziesz mógł drukować co tylko przyjdzie Ci do głowy. Ja w przykładzie pokażę jak można uzupełnić program monitora komputerka AVT-2250 o polecenie drukowania zawartości pamięci operacyjnej w postaci szesnastkowej i znakowej. Funkcja ta może mieć niebanalne znaczenie szczególnie dla ręczniaków, którzy będą mogli “rzucić okiem” na swój cały program wydrukowany na kartce papieru. Oczywiście musicie moi drodzy posiadać choćby najstarszą zdezaktwowaną drukarkę. Ja posłużyłem się leciwą drukarką STAR LC-10, ach lezka kręci się w oku, kiedy wspominam tamte czasy – starych wolnych ale pocziwych pierwszych PC-tów.

Rys.6 Znaczenie sygnałów w trybie 2 pracy





Rys.7 Sposób podłączenia gniazda DB-25/F do układu 8255



Drukarkę taką można nabyć na giełdach w cenach przyprawiających o zawrót głowy, a mianowicie 50,- zł (tak, pięćdziesiąt złotych), mniej niż kosztuje sam komputer.

Przejdźmy zatem do układu.

Aby dołączyć drukarkę do naszego komputerka należy zaopatrzyć się w dodatkowe złącze typu DB-25/F (żeńskie). Następnie zgodnie z rys. 7 wykonujemy połączenia pomiędzy wyprowadzeniami układu 8255 (z rys.2) zgodnie z odpowiadającymi oznaczeniami na obu rysunkach.

Do pracy układu 8255 w charakterze sterownika drukarki wykorzystamy następujące ustawienia konfiguracyjne kostki 8255:

1. port PB ustawimy jako wyjściowy w trybie 1
2. port PA i PCa (PC4...PC7) ustawimy w trybie 0, PA jako wejście, PCa - wyjście
3. wtedy sygnał na linii PC1 (/OBF) użyjemy jako strobujujący dane do drukarki
4. stan na linii PC2 (/ACK) będzie sygnalizował układowi 8255 zakończenie odbioru znaku przez drukarkę
5. nie będziemy korzystać z układu przerwań procesora (przyda się być może do czego innego), dlatego wyjście zgłoszenia przerwania INTRB dołączymy zwrotnie do linii PA7 portu PA (będziemy badać stan tej linii aby wiedzieć kiedy transmisja jest zakończona) dodatkowe linie informacyjne z sygnałami wysyłanymi przez drukarkę dołączymy do wolnych linii portu PA i PC, tak:
  - PA0 do linii /ERROR: sygnał na tej linii przyjmuje stan niski kiedy drukarka nie ma papieru, bądź jest w trybie off-line lub też po prostu jest wyłączona
  - PA1 do linii BUSY: wysoki poziom logiczny na tej linii sygnalizuje jedno ze zdarzeń:
    - a) drukarka jest w stanie off-line,
    - b) lub odłączona od komputerka,
    - c) wewnętrzny bufor drukarki jest pełny,
    - d) drukarka odbiera właśnie znak,
    - e) trwa inicjalizacja drukarki
    - f) wystąpił inny błąd w pracy drukarki
  - PA2 do linii PAPEMPTY: wysoki poziom logiczny na tej linii oznacza wyczerpanie się papieru
  - PA3 do linii SELOUT: wysoki poziom logiczny na tej linii oznacza, że drukarka jest fizycznie połączona z komputerkiem - jest w stanie "on-line"
  - linie PA4...PA6 portu pozostają niewykorzystane.

Pozostało nam podłączenie trzech sygnałów sterujących drukarką

- PC4 do linii AUTOFD: niski poziom podany przez 8255 na to wejście powoduje automatyczne dodawanie znaków w trybie tekstowym przez drukarkę po zakończeniu każdej linii
- PC5 do linii /RESET: niski poziom logiczny uruchamia w pewnych drukarkach procedurę inicjującą
- PC6 do linii SELIN: niski poziom logiczny na tej linii informuje drukarkę że jest ona wybrana i będzie używana. Zwykle linia ta jest połączona z masą, my jednak trzymając się zasady edukacyjnej naszego kursu umożliwimy sobie sterowanie tym sygnałem.

Gotowe! Teraz należy jeszcze dołączyć drukarkę za pomocą typowego kabla drukarkowego z gniazdem DB-25/F, które przed chwilą skrosowaliśmy z wyjściami portów PA, PB i PC układu 8255.

Zapiszmy teraz słowo konfiguracyjne rejestru CTRL układu 8255 dla trybu pracy z drukarką - czyli transmisji synchronicznej gdzie dane są wysyłane z 8255 do drukarki. I tak:

bit 7 = 1 (zawsze jedynka)  
 bit 6 i 5 = 00 (tryb pracy PA jako tryb 0)  
 bit 4 = 1 (linię PA jako wejściową będą odczytywać stany linii drukarki)  
 bit 3 = 0 (linię PC5...PC7 będą sterowały sygnałami AUTOFD, SELIN i /RESET)  
 bit 2 = 1 (tryb pracy rejestru PB - 1)  
 bit 1 = 0 (PB jako wyjście)  
 bit 0 = 1 (aby wolna linia PC3 na wszelki wypadek była wejściem, bo pozostałe linie PC0..PC2 generują sygnały synchronizacji transmisji a więc /ACK, /OBF i INTR)

Wobec tego liczba wpisana do rejestru CTRL przy inicjacji układu 8255 to będzie:

10010101 binarnie, czyli 95h szesnastkowo.

Wykonujemy więc instrukcje:

```
MOV A, #95h
MOV DPTR, #IO_CTRL
MOVX @DPTR, A
```

I gotowe, teraz należy jeszcze odblokować przerzutnik zezwalający na generowanie sygnału przerwania INTRB na linii PC0. Zgodnie z tym co opisałem wcześniej należy wpisać do rejestru CTRL układu 8255 bajt tak jak się ustawia pojedyncze bity rejestru PC. Jak wiemy przerzutnik INTRB znajduje się "pod bitem" D2 rejestru CTRL, dodatkowo ponieważ mamy zamiar ustawić ten bit aby odblokować działanie linii INTRB ustawiamy także bit 0, wobec tego otrzymamy wartość:

00000101 binarnie, czyli 05h szesnastkowo, zapisujemy ją do rejestru CTRL podobnie jak poprzednio:

```
MOV A, #05h
MOV DPTR, #IO_CTRL
MOVX @DPTR, A
Dodatkowo należy zainicjować sygnały sterujące drukarką, czyli linie PC5...PC7 w sposób następujący:
```

```
MOV A, #01010000b ;SELIN=1, RESET=0 (inicjacja), AUTOFD=1
MOV DPTR, #IO_PC
MOVX @DPTR, A
```

a po chwili powinniśmy wykonać instrukcje ustawiające linię /RESET drukarki w stanie nieaktywnym, czyli:

```
MOV A, #01110000b ;SELIN=1, RESET=1 (praca), AUTOFD=1
MOV DPTR, #IO_PC
MOVX @DPTR, A
```

i to już naprawdę wszystko, drukarka jest gotowa do przyjmowania i drukowania znaków.

Teraz trzeba wysłać znak a następnie czekać aż drukarka go wydrukuje i zgłosi ten fakt. Poniżej przedstawiam procedurę drukowania znaku na drukarce dołączonej do komputerka edukacyjnego. Drukowany jest znak z akumulatora Acc.

patrz tabela na następnej stronie

Najpierw utworzyłem zbiór deklaracji adresów układu 8255 dołączonego do komputerka zgodnie z rys.2. Zbiór nazwałem "port8255.inc" i deklarację włączenia go do kompilacji umieściłem w zbiorze w procedurę PRNACC, która wysyła znak do drukarki. Jeżeli transmisja nie powiodła się to procedura wypisuje na displayu napis o błędzie "Err" oraz przed zakończeniem dodatkowo ustawia znacznik C, jeżeli wszystko przebiegło pomyślnie, to procedura kończy się z wyzerowanym znacznikiem C.

Uwaga, procedura ta sprawdza jedynie stan linii INTRB, nie sprawdza stanów linii informacyjnych drukarki /BUSY, PAPEMPTY i SELOUT. Proponuję jako zadanie domowe uzupełnić tę procedurę o detekcję

## Też to potrafisz

Listing 1

```
1 CPU '8052.DEF'
2
Zbior: "const.inc"
Zbior: "bios.inc"

Zbior: "port8255.inc"
1 ,*****
2 ;Deklaracja adresow portow ukkladu 8255
3 ;klasa mikroprocesorowa - odcinek 18, EdW 11/98
4 ,*****
5
6 A000 IO_PA equ A000h
7 A001 IO_PB equ A001h
8 A002 IO_PC equ A002h
9 A003 IO_CTRL equ A003h
10
Zbior: "prnacc.s03"
5
6 0A00 org 0A00h ;offset w obrebie pamieci EPROM komputerka
7
8 ,*****
9 ;* PRNACC * Wysyla znak z akumulatora na drukarke
10 ,*****
11 0A00 C083 prnAcc: push DPH
12 0A02 C082 push DPL
13 0A04 C0E0 push Acc
14 0A06 90A000 chkprn: mov DPTR,#IO_PA ;odczyta stanu drukarki
15 0A09 E0 movx A, DPTR
16 0A0A 540F anl A,#0Fh
17 0A0C 6409 xrl A,#1001b ;czy drukarka gotowa ?
18 0A0E 6017 jz prnok ;tak to drukuj
19 0A10 prnerror:
20 0A10 120274 lcall CLS
21 0A13 757879 mov DL1,#_E
22 0A16 757950 mov DL2,#_r
23 0A19 757A50 mov DL3,#_r
24 0A1C 1202C5 lcall CONIN ;czekanie na dowolny klawisz
25 0A1F D3 setb C ;ustawienie znacznika błedu (C=1)
26 0A20 D0E0 pop Acc
27 0A22 D082 exit: pop DPL ;i zakonczenie drukowania
28 0A24 D083 pop DPH
29 0A26 22 ret
30 0A27 D0E0 prnok: pop Acc ;od tej instrukcji gdy OK !
31 0A29 90A001 mov DPTR,#IO_PB
32 0A2C F0 movx DPTR,A ;wyslanie znaku do drukarki
33 0A2D 90A000 mov DPTR,#IO_PA ;odczyt stanu drukarki
34 0A30 E0 wait: movx A, DPTR
35 0A31 5480 anl A,#80h ;czy mozna wyslac nastepny znak ?
36 0A33 60FB jz wait ;nie to czekaj
37 0A35 C3 clr C ;zakonczenie drukowania OK. !
38 0A36 80EA sjmp exit
39
40 0A38 END

Kompilacja zakonczona pomyslnie !
Zbior: "prnacc.s03" , 56 bajt(ow), 0.1 sekund(y).
```

tych sygnałów i spróbować w domowym zaciszu pobawić się z tymi sygnałami.

Procedura została skompilowana z offsetem 0A00h, czyli w obrębie pamięci programu komputerka. Jeżeli ktoś chce, to może zmodyfikować pamięć monitora dodając na stałe procedurę PRNACC do pamięci programu EPROM. Sposób modyfikacji przedstawilem we wrześniowym numerze EdW. W każdym razie trzeba dysponować programatorem pamięci EPROM.

Jeżeli nie posiadasz programatora to procedurę PRNACC powinieneś wykorzystywać w aplikacjach pisanych przez siebie, tzn. umieszczać listing podany powyżej w pliku źródłowym twego programu (oczywiście pomijając dodatkowe oznaczenia i informacje wygenerowane przez kompilator PASM51.EXE).

Użycie przedstawionej procedury jest bardzo proste i wyglądać może właśnie tak:

```
MOV A, #'A' ;wydrukuj literę A
LCALL PRNACC
JNC nastepny_znak
blad_drukarki:
.....
.....;instrukcje w wypadku błedu drukowania
```

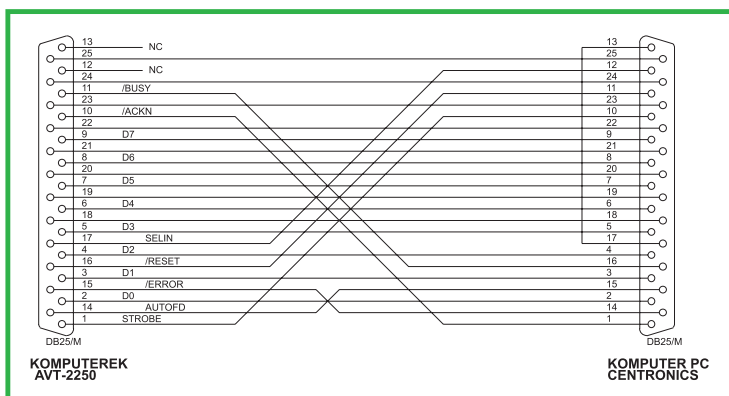
Na deser proponuję krótką procedurę dodatkową, która korzysta z procedury PRNACC i drukuje tekst (ciąg bajtów) zakończony znakiem #0 (zero) zaczynający się od adresu podanego w DPTR. Oto ona:

Teraz możesz np. w swoim programie testującym spróbować wydrukować zdanie:

Listing 2

```
CPU '8052.def'
PRNACC equ 0A00h

,*****
;* PRNTEXT * Wysyla tekst ASCIIz na drukarke (dane: mov DPTR,#tekst)
,*****
PRNTEXT:
 movx A, DPTR ;pobranie znaku z bufora
 cjne A,#0,ok1 ;czy znak konca tekstu ?
 ret
 ;tak to koniec procedury
 lcall PRNACC ;nie to wydrukuj znak
 jnc ok2 ;czy blad drukowania ?
 ret
 ;tak to zakoncz procedure
 inc DPTR ;nie to nastepny znak
 sjmp prntxt
 ret
```



Rys.8 Schemat kabla do połączenia komputera PC z AVT-2250

“Hello z systemu AVT-2250!” za pomocą sekwencji instrukcji:

```
MOV DPTR, #zdanie
LCALL PRNTXT
.....
.....
zdanie db 'Hello z systemu AVT-2250!', 0
```

Pamiętaj aby na końcu zawsze postawić znak #0, w przeciwnym przypadku procedura pójdzie w przysłowiowe “maliny”.

## SYMULOWANIE DRUKARKI, CZYLI JAK ODBIERAĆ ZNAKI Z ZEWNĄTRZ

Uff, wiesz już drogi czytelniku w jaki sposób wysłać znaki na drukarkę za pomocą układu 8255. Czy nie warto byłoby także móc ich odbierać, ale skąd?, no np. z zewnętrznego urządzenia wyposażonego w port równoległy kompatybilny z naszym, chociażby z pocziwego PC’ta lub innego komputera, jaki posiadasz.

W tej części artykułu pokażę w jaki sposób komputer edukacyjny AVT-2250 może bez żadnych przeróbek zmontowanego już układu 8255 (zgodnie z rys.2) udawać drukarkę i przyjmować znaki z zewnątrz. Jak się pewnie domyślasz do zrealizowania tego celu wykorzystam ten sam tryb pracy portu jak poprzednio (tryb 1) lecz port PB będzie pracował jako wejście danych. Sytuację tę możesz zobaczyć na rys. 4 w tej jego części gdzie pokazany jest port PB w trybie 1 oraz towarzyszące mu sygnały synchronizujące transmisję, czyli /STR, INB i INTRB.

Pierwszym krokiem jest ustalenie składni słowa konfiguracyjnego portu CTRL układu 8255. Wszystkie bity tego słowa będą takie same jak poprzednio za wyjątkiem dwóch najmłodszych, oto one:

bit 1 = 1 ( PB jako wejście )  
bit 0 = 0 ( Pcb jako wyjście )

Listing 3

```
7 0B00 org 0B00h
8
9 ;*****
10 ;* INPACC * Czekaj na znak z portu rownoleglego i do Acc
11 ;*****
12 0B00 C083 INPACC: push DPH
13 0B02 C082 push DPL
14 0B04 90A000 mov DPTR,#IO_PA ;czy układ 8255 zajety ?
15 0B07 E0 waitp: movx A, DPTR
16 0B08 5480 anl A,#80h
17 0B0A 60FB jz waitp ;tak to poczekaj
18 0B0C 90A001 mov DPTR,#IO_PB ;nie to odczytaj dana
19 0B0F E0 movx A, DPTR ;dana w Acc
20 0B10 D082 pop DPL
21 0B12 D083 pop DPH
22 0B14 22 ret
23
24 0B15 END
```

Kompilacja zakończona pomyślnie !  
Zbiór: "inpacc.s03", 21 bajt(ow), 0.1 sekund(y).

stąd więc otrzymamy słowo: 10010110 binarnie, czyli 96h szesnastkowo. Podobnie jak poprzednio trzeba wysłać jest do portu 8255, właśnie tak:

```
MOV A, #96h
MOV DPTR, #IO_CTRL
MOVX @DPTR, A
```

Pozostał jeszcze do odblokowania przerzutnik zgłoszenia przerwania INTRB tak więc do portu układu 8255 wysyłamy taką samą wartość jak w przypadku obsługi drukarki, czyli:

```
MOV A, #05h
MOV DPTR, #IO_CTRL
MOVX @DPTR, A
```

Następnie należy wykonać specjalny kabel do połączenia z urządzeniem zewnętrznym, w naszym przypadku będzie to komputer PC z równoległym złączem typu Centronics (drukarkowym). Rys.8 przedstawia schemat połączeń kabla do transmisji z komputerem PC. Oba wtyki po dwóch stronach kabla będą typu męskiego DB-25/M, lecz ze względu na wykonane połączenia należy zaznaczyć (flamastrem) tę stronę którą wtykamy do komputera PC a drugą, jako dołączaną do portu 8255 komputerka AVT-2250. Podczas wykonywania kabla radzę Ci zastanowić się na przyporządkowaniu poszczególnych linii sygnałom portu Centronics komputera PC. Dla ułatwienia powiem tylko że dzięki skrzyżowaniom sygnałów możliwe jest połączenia następujących kompatybilnych par sygnałów:

| AVT-2250 | ↔ | PC     |
|----------|---|--------|
| STROBE   | z | ACK    |
| BUSY     | z | RESET  |
| ERROR    | z | AUTOFD |

Jeżeli kabel został wykonany to można zabrać się do zainicjowania stanów na odpowiednich liniach łącza, tak aby komputer PC zaczął “widzieć” nasz komputer i aby można było wysłać do niego jakieś znaki.

W tym celu do portu PC należy wysłać bajt: 00100000b, a to dlatego, że wtedy ustawimy sygnały:

PAPER EMPTY (OK.), BUSY i ERROR, tak że komputer będzie widziany przez PC’ta jak potencjalna gotowa do pracy drukarka. Wykonujemy więc instrukcje:

```
MOV A, #00100000b ;PAPER OK, BUSY, not ERROR
MOV DPTR, #IO_PC
MOVX @DPTR, A
```

i układ jest gotowy do odbierania znaków z urządzenia zewnętrznego.

Poniżej zamieszczam uproszczoną procedurę oczekiwania na znak z łącza równoległego i umieszczenie go w akumulatorze. Jako pracę domową proponuję uzupełnienie tej procedury funkcją przeterminowania, tak aby komputer edukacyjny przy braku transmisji po prostu nie zawiesił się.  
patrz tabela na następnej stronie

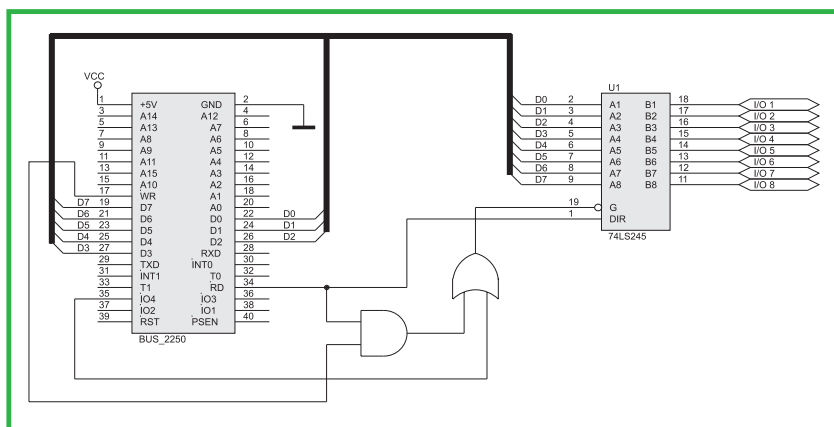
Procedura została skompilowana z off-setem 0B00h, tak aby znaleźć się w obszarze pamięci programu monitora komputera, a jednocześnie nie kolidować z wcześniejszą procedurą obsługi drukarki. Tak ja poprzednio procedury tej można także używać w swoich aplikacjach wstawiając ją do zbioru źródłowego.

Korzystanie z procedury INPACC jest proste i sprowadza się do jej wywołania:

```
LCALL INPACC
```

.....  
;a co dalej zrobisz z tym znakiem to twoja sprawa

## Też to potrafisz



Rys.1 Rozwiązanie zadania nr 1

To na razie wszystko, co przygotowałem na dzisiejszy odcinek klasy mikroprocesorowej. Myślę że spora ilość informacji do przemyślenia a także sprawdzenia w praktyce umili Ci, drogi Czytelniku jesienne chłodne wieczory. Powodzenia.

Sławomir Surowiński

## LEKCJA 11

Rozwiązania zadań z poprzedniej lekcji przedstawione są na rysunkach poniżej. I tak rysunek 1 przedstawia zmodyfikowany z rys.9 w poprzednim numerze układ uwzględniający sygnał /WR procesora.

Na rys.2 pokazałem jak połączyć dwa układy 74LS138 tak aby zastąpić jeden demultiplexer 74LS154.

Wreszcie na rys.3 znajduje się zmodyfikowany układ dekodera z rys.10 (z poprzedniego numeru EdW), dzięki któremu układ U1 zapisywany jest pod adresem A000h a odczytywany pod adresem A001h. Sądzę że analiza powyższych schematów nie nastręczy nikomu trudności. Jako zadanie na kolejny raz proponuję zapisanie procedury drukowania zawartości zewnętrznej pamięci danych komputerka AVT-2250, w zadany z klawiatury zakresie np. od adresu 8000h do 80FFh w postaci szesnastkowej i znakowej według przykładowego szablonu:

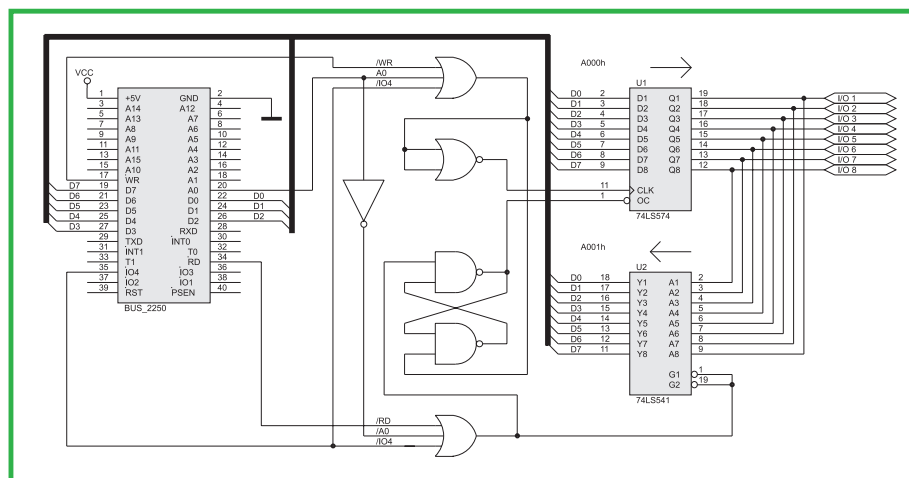
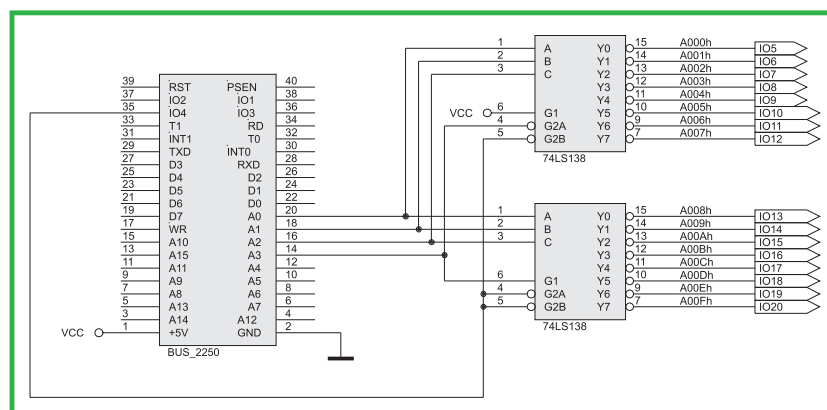
Adres - 16 bajtów kolejnych danych w HEX - te same bajty jako znaki

Przykładowy wydruk powinien wyglądać jak ten na dole strony

Wydruk przedstawia zawartość pamięci o podanych adresach z zakresu: 2600h... 26FFh. Wesołej zabawy i do zobaczenia w następnym numerze EdW, gdzie znajdzie się rozwiązanie powyższego zadania.

Sławomir Surowiński

Rys.2 Rozwiązanie zadania nr 2



Rys.3 Rozwiązanie zadania nr 3

| Address | Data                                            | Character         |
|---------|-------------------------------------------------|-------------------|
| (2600h) | 0E 57 31 C0 50 9A E2 36 7D 0E 9A DF 35 7D 0E 80 | .W1.P.6)...5)..   |
| (2610h) | 3E CA 8A 00 74 06 C6 46 FB 01 EB 04 C6 46 FB 02 | >...t.F....F..    |
| (2620h) | 8A 46 FB 50 B0 0F 50 9A AA 10 CF 0C 8D 7E FB 16 | .F.P.P.....~.     |
| (2630h) | 57 B0 02 50 E8 05 F8 8D 7E FC 16 57 9A 1F 0C CF | W..P.....W....    |
| (2640h) | 0C 80 3E 7E 89 00 74 07 C6 06 7E 89 00 EB 17 8A | ...~.t...~.....   |
| (2650h) | 46 FB 3C 01 75 07 C6 06 CA 8A 01 EB 09 3C 02 75 | F.<u.....<u       |
| (2660h) | 05 C6 06 CA 8A 00 89 EC 5D C3 55 89 E5 31 C0 9A | .....].U..1..     |
| (2670h) | 30 05 7D 0E BF 98 89 1E 57 B8 B7 00 50 9A E3 37 | 0.].....W...P..7  |
| (2680h) | 7D 0E BF 98 89 1E 57 BF 18 8A 1E 57 9A 96 38 7D | ].....W....W.8}   |
| (2690h) | 0E 83 C4 04 BF 98 89 1E 57 9A 5B 38 7D 0E 9A 47 | .....W.[8]..G     |
| (26A0h) | 0C CF 0C B0 07 50 E8 8F FA 31 C0 9A 16 01 7D 0E | .....P....1....]. |
| (26B0h) | 5D C3 00 55 89 E5 31 C0 9A 30 05 7D 0E 80 3E C8 | ]..U..1..0.]...>. |
| (26C0h) | 8A 00 B0 00 75 01 40 A2 C8 8A BF 94 89 1E 57 9A | ....u. ....W.     |
| (26D0h) | 1F 0C CF 0C C6 06 7B 89 00 C6 06 7E 89 01 C6 06 | .....[....~....   |
| (26E0h) | 52 90 FF B0 00 50 BF 32 0C 0E 57 E8 7C FB 5D C3 | R....P2..W.I.]    |
| (26F0h) | 55 89 E5 31 C0 9A 30 05 7D 0E A0 CB 8A 30 E4 48 | U..1..0.]....0.H  |



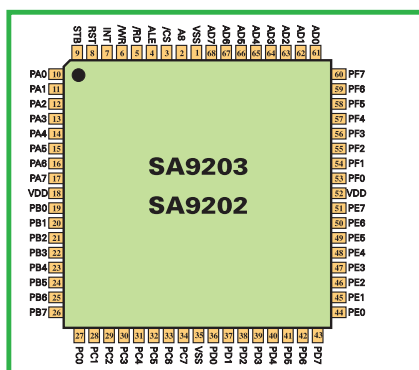
W tym odcinku kończymy omawianie układów I/O ciekawym przykładem wykorzystania dostępnego ostatnio na rynku polskim układu scalonego SA9203. Kostka może stać się o tyle atrakcyjna gdyż jest ostatnio dostępna w sieci handlowej AVT, toteż każdy z Was kto zainteresuje się niniejszym układem, będzie mógł spróbować wykorzystać go dla swoich potrzeb – a jak się za chwilę przekonacie zastosowań może być wiele. Dziś także rozwiązanie zadania z poprzedniej lekcji – czyli drukowanie z komputerka AVT-2250 na zwykłej drukarce komputerowej wyposażonej w równoległe złącze standardu Centronics.



Dziś chciałbym zapoznać Cię drogi czytelniku z ciekawym układem I/O, którego możliwości odpowiadają mniej więcej dwóm, omawianym wcześniej układom 8255. Chodzi o układ dość egzotycznej bo południowoafrykańskiej firmy produkującej szeroka gamę układów dla potrzeb nowoczesnej telekomunikacji - SAMES. Mowa będzie o układzie SA9203. Ponieważ od niedawna układ ten jest dostępny w ofercie handlowej AVT, a jego cena jest przystępna jak na możliwości kostki, postanowiłem zakończyć cykl o układach I/O omówieniem właśnie jego możliwości. Drugim powodem dla którego chcę zwrócić szczególną uwagę na kostkę SA9203 to idealna wprost kompatybilność z rodziną mikroprocesorów 8051 jeżeli chodzi o połączenie obu tych układów. Dzięki zastosowaniu SA9203 system mikroprocesorowy może zostać wzbogacony o 6 dodatkowych uniwersalnych portów wejścia-wyjścia, każdy o szerokości 8-miu bitów. W sumie daje to aż 48! dodatkowych końcówek do dowolnego wykorzystania.

## UKŁAD SA9203 – TROCHĘ TEORII

Kostka została wykonana w technologii CMOS. Jak wcześniej powiedziałem, układ świetnie nadaje się do podłączenia z mikrokontrolerem 8051 a to ze względu na fakt posiadania multiplexowanej szyny danych i adresu. Układ dostarczany jest przez producenta w typowej obudowie PLCC68. Nie powinno to jednak odstraszyć od zastosowania nawet amatora elektroniki, bowiem do układu można zastosować odpowiednią podstawkę która posiada typowy całowy rozstaw (2,54 mm). Na rys.1 przedstawiono opis wyprowadzeń układu a na rys.2 schemat wewnętrzny układu SA9203.



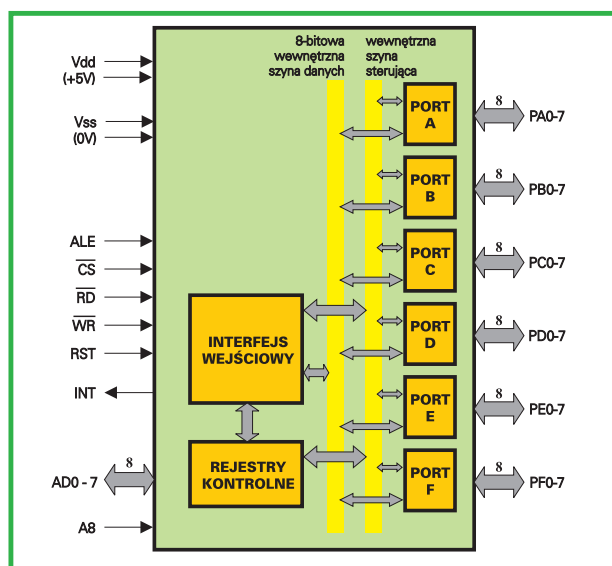
Rys.1 Rozkład wyprowadzeń układu SA9203.

Układ składa się z: interfejsu wejściowego, łączącej multiplexowaną szynę danych / adresu z ze-

wnętrznym układem sterującym, rejestrów konfiguracyjnych oraz rejestrów 6-ciu portów A...F. Znaczenie zewnętrznych sygnałów sterujących podaje w tabeli 1.

W tabeli 2 przedstawiam parametry elektryczne układu SA9203. Jak widać parametry odpowiadają warunkom zasilania TTL, czyli 5V. Układ SA9203 może z powodzeniem pracować z układami wykonanymi w technologii CMOS, a także TTL-LS, lub nawet TTL przy zachowaniu odpowiednich dopuszczalnych obciążeń.

Na rys.3 przedstawiono układ i adresy wewnętrznych rejestrów układu. Układ podobnie jak mikroprocesor 8051 posiada 8-bitową multiplexowaną szynę adresową – danych AD0...AD7, co dla zewnętrznego układu sterującego kostką SA9203 zajmuje obszar 256 bajtów w jego (8051) przestrzeni adresowej. Wybrane obszary układu zajęte są przez



Rys.2 Schemat wewnętrzny układu SA9203.

## Też to potrafisz

| Tabela 1 |       |           |                                                                                                                                                                                                                                                                 |
|----------|-------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pin      | Typ   | Symbol    | Opis                                                                                                                                                                                                                                                            |
| 18,52    |       | VDD       | zasilanie układu +5V;                                                                                                                                                                                                                                           |
| 1,35     |       | VSS       | masa zasilania 0V                                                                                                                                                                                                                                               |
| 61...68  | we/wy | AD0...AD7 | 3-stanowa multipleksowana szyna danych/adresu. 8-bitowy adres zatraskiwany jest we wnętrzu układu SA9203 podczas opadającego zbocza sygnału ALE. Dane zapisywane są lub odczytywane po podaniu odpowiednio sygnałów /WR (zapisu) lub /RD (odczytu);             |
| 2        | we    | A8        | nie używany w układzie SA9203, powinien być połączony z masą (Vss), znaczenie tego pinu opiszę w dalszej części artykułu;                                                                                                                                       |
| 3        | we    | /CS       | aktywny niski sygnał na tym wejściu powoduje wybór układu;                                                                                                                                                                                                      |
| 4        | we    | ALE       | pin kontrolujący zatrzaśnięcie adresu na szynie AD0...AD7 podczas zapisu przez urządzenie zewnętrzne, następuje to podczas opadającego zbocza tego sygnału;                                                                                                     |
| 5        | we    | /RD       | poziom niski na tym wejściu pozwala na odczyt wewnętrznych rejestrów układu                                                                                                                                                                                     |
| 6        | we    | /WR       | poziom niski na tym wejściu powoduje zapis danej do wewnętrznego rejestru układu                                                                                                                                                                                |
| 7        | wy    | INT       | programowane wyjście zgłoszenia przerwania do układu zewnętrznego, możliwość ustalenia polaryzacji oraz uaktywnienia tego pinu                                                                                                                                  |
| 8        | we    | RST       | wysoki poziom podany na to wejście powoduje zresetowanie układu                                                                                                                                                                                                 |
| 9        | we    | STB       | wejście zatraskiwania danej w porcie A, gdy port ten pracuje jako wejście,                                                                                                                                                                                      |
| 10...17  | we/wy | PA0...PA7 | uniwersalny 8-bitowy port I/O. Możliwość indywidualnego zdefiniowania każdego pinu portu jako zatraskiwanego wyjścia lub wejścia. W trybie pracy jako wejście port może pracować w trybie zatraskiwania (sygnałem STB) lub jako "przezroczysty" ("transparent") |
| 19...26  | we/wy | PB0...PB7 | 8-bitowy uniwersalny port I/O. Wszystkie piny mogą być ustawione jako zatraskiwane wyjścia lub jako wejścia typu "transparent".                                                                                                                                 |
| 27...34  | we/wy | PC0...PC7 | identyczny jak port B                                                                                                                                                                                                                                           |
| 36...43  | we/wy | PD0...PD7 | identyczny jak port B                                                                                                                                                                                                                                           |
| 44...51  | we/wy | PE0...PE7 | identyczny jak port B                                                                                                                                                                                                                                           |
| 53...60  | we/wy | PF0...PF7 | identyczny jak port B                                                                                                                                                                                                                                           |

porty konfiguracyjne oraz porty PA...PF jak pokazano na rysunku. Układ pozwala na dwójakie adresowanie każdego z rejestrów portów PA...PF. Pierwszy polega na jednoczesnym adresowaniu całego portu, drugi pozwala na zaadresowanie pojedynczego pinu każdego z portów. W tym drugim przypadku przy odczycie danej w postaci bajtu, siedem najstarszych bitów nie ma znaczenia, jedynie najmłodszy D0 wskazuje na stan pinu lub wymusza go w przypadku pracy portu jako cyfrowego wyjścia.

Dodatkowe rejestry sterujące pracą całego układu zawsze adresowane są bajtowo. Dość użyteczną funkcją w przypadku tych rejestrów jest możliwość odczytu ich zawartości. Dzięki temu programista nie musi zapamiętywać ich stanu po zapisie w dodatkowych zmiennych wykorzystywanych w programie.

Na rys.4 przedstawiona jest mapa adresowa poszczególnych portów w trybie adresowania bitowego.

| Tabela 2                       |        |      |      |      |           |                       |
|--------------------------------|--------|------|------|------|-----------|-----------------------|
| Parametr                       | Symbol | Min  | Typ  | Max  | Jednostka | Warunek pomiaru       |
| zasilanie                      | Vdd    | 4,75 | 5,0  | 5,25 | V         |                       |
| pobór prądu (statyczny)        | Idds   |      | 15   | 50   | uA        | Vdd = 5,0V            |
| pobór prądu (dynamiczny)       | Iddd   |      |      | 20   | mA        | Vdd = 5,0V            |
| napięcie wej. w stanie wysokim | Vih    | 2,0  |      |      | V         | Vdd = 5V              |
| napięcie wej. w stanie niskim  | Vil    |      |      | 1,0  | V         | Vdd = 5V              |
| napięcie wyj. w stanie wysokim | Voh    | 4,5  | 4,7  |      | V         | Vdd = 5V<br>Ioh = 5mA |
| napięcie wyj. w stanie niskim  | Vol    |      | 0,25 | 0,5  | V         | Vdd = 5V<br>Ioh = 7mA |

| MNEMONIK | REJESTR                                       | ADRES W TRYBIE |           |
|----------|-----------------------------------------------|----------------|-----------|
|          |                                               | BAJTOWYM       | BITOWYM   |
| PA       | PORT A                                        | 00h            | 00h...07h |
| PB       | PORT B                                        | 08h            | 08h...0Fh |
| PC       | PORT C                                        | 10h            | 10h...17h |
| PD       | PORT D                                        | 18h            | 18h...1Fh |
| PE       | PORT E                                        | 20h            | 20h...27h |
| PF       | PORT F                                        | 28h            | 28h...2Fh |
| PACR     | REJESTR KONTROLNY PORTU A                     | 70h            | 70h       |
| PAICR    | REJESTR KONFIGURACJI PORTU A W TRYBIE WEJŚCIA | 71h            | 71h       |
| IOCR     | REJESTR KONFIGURACJI PORTÓW B...F             | 72h            | 72h       |
| PAMR     | REJESTR TRYBU ADRESOWANIA PORTÓW              | 73h            | 73h       |

Rys.3 Rejestry wewnętrzne układu SA9203

|        |      |      |      |      |      |      |      |      |
|--------|------|------|------|------|------|------|------|------|
| PORT A | 07   | 06   | 05   | 04   | 03   | 02   | 01   | 00   |
|        | BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |
| PORT B | 0F   | 0E   | 0D   | 0C   | 0B   | 0A   | 09   | 08   |
|        | BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |
| PORT C | 17   | 16   | 15   | 14   | 13   | 12   | 11   | 10   |
|        | BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |
| PORT D | 1F   | 1E   | 1D   | 1C   | 1B   | 1A   | 19   | 18   |
|        | BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |
| PORT E | 27   | 26   | 25   | 24   | 23   | 22   | 21   | 20   |
|        | BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |
| PORT F | 2F   | 2E   | 2D   | 2C   | 2B   | 2A   | 29   | 28   |
|        | BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |

Rys.4 Mapa portów przy adresowaniu bitowym.

Poniżej zapoznam cię ze znaczeniem poszczególnych portów kontrolnych układu SA9203.

PACR - rejestr kontrolny portu A. Na rys.5 pokazano znaczenie poszczególnych bitów rejestru. Odpo-

|                  |                  |                  |                  |                  |                  |                  |                  |
|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| BIT7             | BIT6             | BIT5             | BIT4             | BIT3             | BIT2             | BIT1             | BIT0             |
| PAD <sub>7</sub> | PAD <sub>6</sub> | PAD <sub>5</sub> | PAD <sub>4</sub> | PAD <sub>3</sub> | PAD <sub>2</sub> | PAD <sub>1</sub> | PAD <sub>0</sub> |

Kierunek PA<sub>0-7</sub>  
0 = wejście  
1 = wyjście

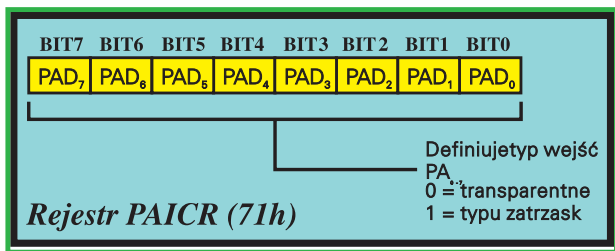
**Rejestr PACR (70h)**

Rys.5 Rejestr specjalny PACR

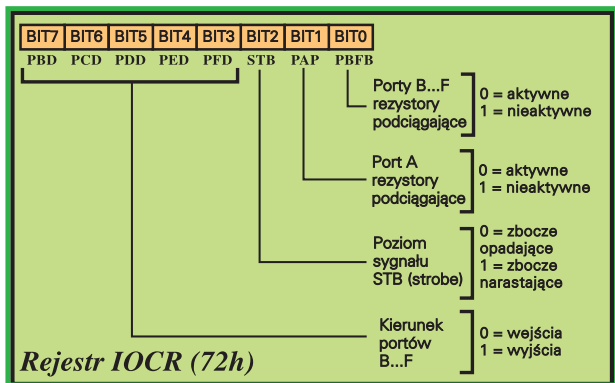
wiednie ustawienie lub wyzerowanie pozycji pozwala na indywidualne ustalenie pinu portu A jako wejścia lub wyjścia cyfrowego.

PAICR - rejestr konfiguracyjny portu A w trybie wejścia. Rys.6 przedstawia znaczenie bitów rejestru. W przypadku ustawienia pinu jako zatraskiwanego wejścia ("latched input") dane zostają zapamiętane po nadejściu sygnału na wejściu STB. W przypadku ustawienia dowolnego pinu portu A jako wyjścia, odpowiadający mu bit w rejestrze PAICR nie ma wpływu na działanie tej linii portu.

IOCR - rejestr konfiguracyjny wejścia-wyjścia. Rejestr pozwala na konfigurację



### Rys.6 Rejestr specjalny PAICR



Rys.7 Rejestr konfiguracyjny IOCR.

portów PB...PF, załączanie wewnętrznych rezystorów podciągających w trybie wejścia, oraz określa polaryzację sygnału STB (patrz rys.7).

Jak wspominałem wcześniej wszystkie wyprowadzenia portów układu SA9203 posiadają wbudowane rezystory podciągające pin portu kiedy ten pracuje jako wejście. Zwalnia to użytkownika od stosowania dodatkowych elementów rezystancyjnych w niektórych aplikacjach. Rezystory te mogą być uaktywnione lub wyłączone oddzielnie dla każdego z portów PA...PF

Bit IOCR.2 definiuje polaryzację sygnału STB ("strobe") który zatrzaskuje daną w porcie A (lub wybranych jego pinach) kiedy ten pracuje jako wejście. W przypadku ustawienia bitu na 0, dane zatrzaskiwane są podczas opadającego zbocza sygnału STB, gdy bit = 1, podczas narastającego zbocza sygnału.

PAMR - rejestr trybu adresowania. Zgodnie z rys.8 ustawienie lub wyzerowanie odpowiedniego bitu w tym rejestrze pozwala na zmianę trybu adresowania poszczególnych portów A...F, aktywację oraz polaryzację sygnału zgłoszenia przerwania INT.

INT - wyjście zgłoszenia

przerwania w przypadku za-  
trażenia danej w porcie A  
po nadejściu sygnału STB.  
Aktywacja pinu następuje  
po odpowiednim ustawie-  
niu bitu PAMR.1. Polaryza-  
cja zgłoszenia przerwania  
ustalana jest poprzez bit  
PAMR.0.

RST - podanie wysokiego poziomu na to wejście resetuje cały układ. Zawartość wszystkich rejestrów zostaje wyzerowana. Następstwem tego jest:

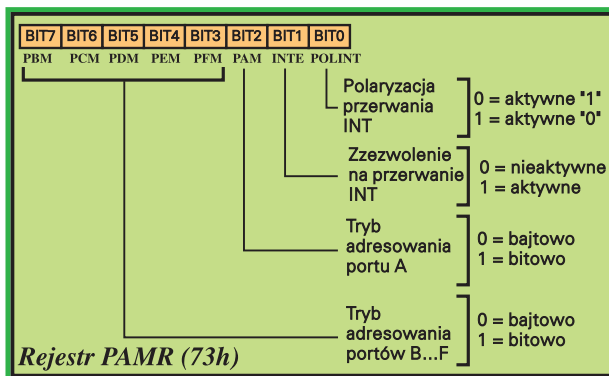
- wszystkie piny portów PA...PF ustawione zostają jako wejścia

- wejścia portu A pracują jako transparentne
- rezystory podciągające są uaktywnione we wszystkich portach

- wyjście przerwania jest nieaktywne

- adresowanie portów ustalone zostaje jako bajtowe

Aktywny (wysoki) stan sygnału RST powinien trwać minimum 100ns.



Rys.8 Rejestr adresowania PAMR.

## PRAKTYCZNE UKŁADY Z SA9203

Nie przedłużając teoretycznych wywodów na temat układu SA9203 na rys.9 przedstawiam najprostszy sposób na połączenie z mikroprocesorem serii '51 w wersji z wewnętrzną pamięcią programu.

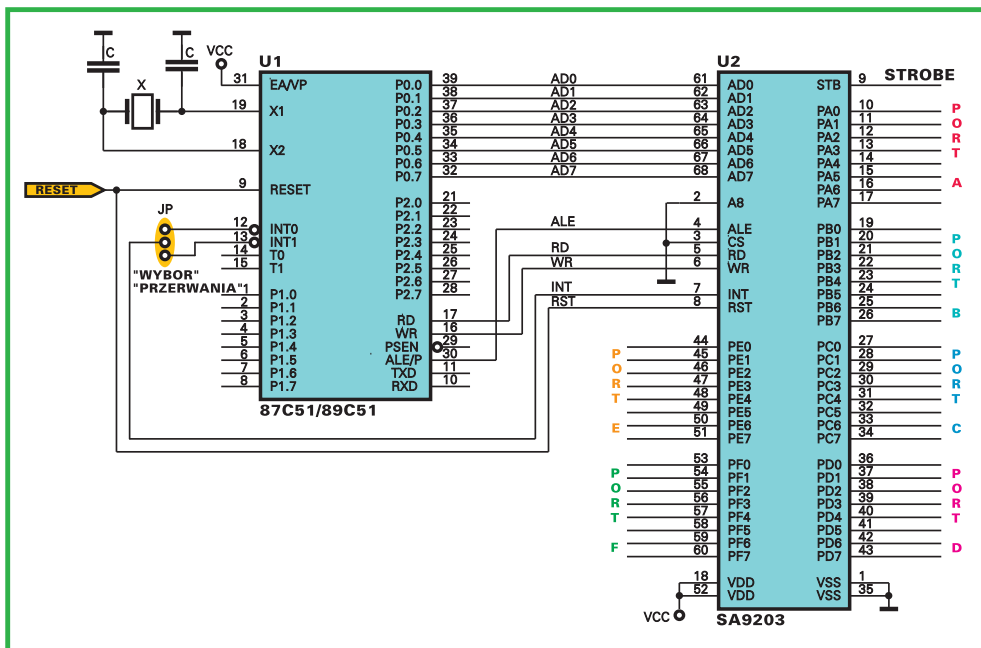
Jak widać z rysunku, do działania nie są potrzebne żadne dodatkowe układy sterujące czy dekodery adresów. Dzięki wspomnianemu wcześniej kompatybilnej z 8051, magistrali adresowej-danych układ SA9203 "widziany" jest przez procesor jako 256 komórek w zewnętrznej pamięci danych, do której procesor odwołuje się za pośrednictwem rozkazów:

MOVX @DPTR, A      podczas zapisu do portu

oraz

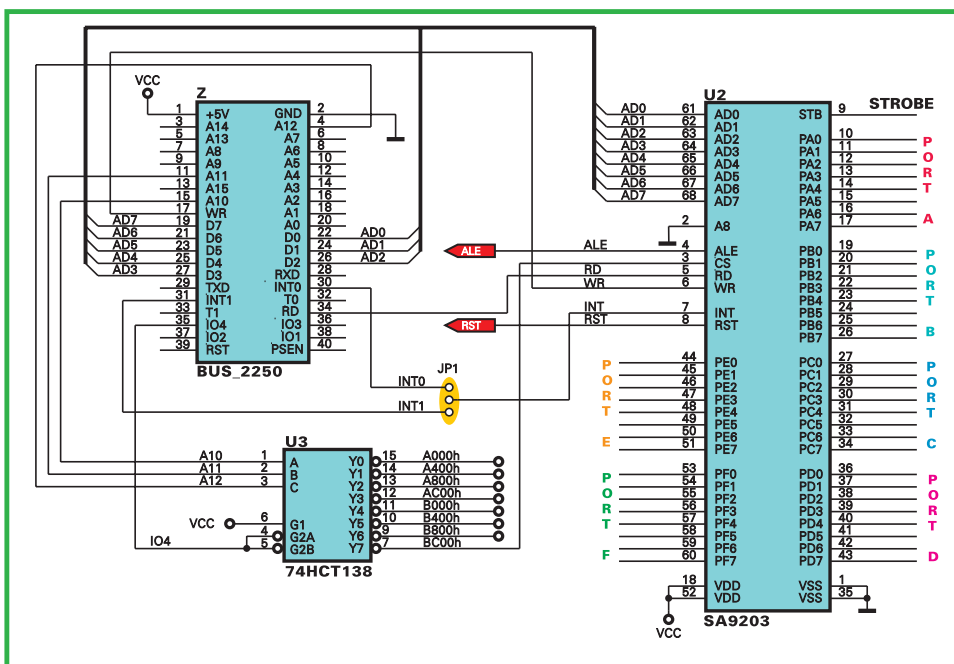
MOVX A, @DPTR      przy odczycie z portu

Tak naprawdę, to możliwe jest zaadresowanie (zgodnie z rys.3) tylko do adresu 73h, pod którym znajduje się rejestr PAMR. Pozostałe komórki są niewykorzystane. Ktoś może zapytać, po co w układzie SA9203 potrzebna jest linia adresowa A8 ? Otóż istnieje wersja układu SA9202, kompatybilna z omawianym SA9203, lecz posiadająca dodatkowo w swojej strukturze 256 bajtów statycznej pamięci RAM. Tak więc zaadresowanie układu SA9202 w zakresie 100h...1FFh (linia A8=1) powoduje dostęp do tej właśnie pamięci. Jeżeli ktoś z czytelników będzie miał okazję nabyć tę wersję układu, zachęcam do zakupu i eksperymentów. Dodatkowa pamięć RAM idealnie może nadawać się do przechowywania danych podczas wykonywania jakiegoś programu. Zintegrowanie jej w układzie portu SA9202 zwalnia od stosowania dodatkowej kostki SRAM, co ma wpływ na wielkość i skomplikowanie obwodu drukowanego.



Rys.9 Współpraca układu SA9203 z kontrolerem 87C51/89C51

## Też to potrafisz



Rys.10 Sposób na dołączenie SA9203 do komputerka edukacyjnego.

W przypadku chęci dołączenia układu SA9203 do komputerka AVT-2250 najlepiej posłużyć się schematem z rys.10.

W układzie oprócz SA9203 zastosowano dodatkowe dekodowanie adresów dzięki wykorzystaniu dekodera 1 z 8-miu typu 74HCT138 (74LS138). Użycie tego układu wydało mi się uzasadnione, a to ze względu na poprzednio prezentowane przykłady. Jak zapewne pamiętasz wszystkie one korzystały z sygnału magistrali komputerka IO4. Ponieważ możesz zastosować ich kilka za jednym razem, należało zastosować dodatkowe dekodowanie adresów, tak aby podzielić obszar adresowy dekodowany przez sygnał IO4 na kilka mniejszych (w tym przypadku na osiem 1kbajtowych). W naszym przykładzie z rys.10 układ SA9203 dekodowany jest przez sygnał Y7 kostki U3, co odpowiada adresom BC00h...BFFFh. Ponieważ na złącze BUS\_2250 komputerka nie wyprowadzono sygnałów ALE oraz RESET (polaryzacja dodatnia), należy te dwa połączenia wykonać bezpośrednio łącząc te końcówki kawałkiem drutu.

Aby teraz "dobrać" się do układu SA9203 należy w programie jego obsługi zadeklarować następująco:

```
SA9203 EQU BC00h
```

Ze względu na to że układ zawiera kilka rejestrów wewnętrznych, które opisałem wcześniej w programie warto od razu zapisać kolejne deklaracje:

```
PORTA EQU BC00h ;adres portu PA
PORTB EQU BC08h ;adres portu PB
PORTC EQU BC10h ;adres portu PC
PORTD EQU BC18h ;adres portu PD
PORTE EQU BC20h ;adres portu PE
PORTF EQU BC28h ;adres portu PF
PACR EQU BC70h ;adres rejestru kontrolnego portu PA
PAICR EQU BC71h ;adres rej. konfiguracji portu PA
IOCR EQU BC72h ;adres rej. konfiguracji wejść-wyjść
PAMR EQU BC73h ;adres rejestru wyboru trybu adresowania
```

Można od razu zadeklarować adresy poszczególnych bitów portów, które przydadzą się w trybie adresowania bitowego, np. tak:

```
BITPA0 EQU BC00h ;adres pinu 10 układu (PA0)
BITPA1 EQU BC01h
BITPA2 EQU BC02h
BITPA3 EQU BC03h
BITPA4 EQU BC04h
BITPA5 EQU BC05h
BITPA6 EQU BC06h
BITPA7 EQU BC07h ;adres pinu 17 układu (PA7)
BITPB0 EQU BC08h
BITPB7 EQU BC0Fh
```

i tak dalej aż do portu PF gdzie deklaracje będą wyglądały jak następuje:

```
BITPF0 EQU BC28h
BITPF7 EQU BC2Fh
```

Teraz można zabrać się do programowania układu. W naszym pierwszym przykładzie zaprogramujemy układ SA9203 do pracy w trybie wyjścia – wszystkie porty PA...PF ustawimy jako wyjścia.

Najpierw zajmiemy się rejestrem PACR – konfiguracji portu PA. Ponieważ port PA ma pracować jako wyjście zgodnie z rys.5 należy ustawić wszystkie bity tego rejestru, tak więc piszemy instrukcje:

```
MOV A, #0FFh
MOV DPTR, #PACR
MOVB @DPTR, A
```

Ponieważ port PA ma pracować jako wyjście programowanie rejestru PAICR jest zbędne. Pora teraz na rejestr IOCR, gdzie należy ustawić bity 7...3 tak aby porty PB...PF pracowały

ty jako wyjścia. Pozostałe bity rejestru są w trybie wyjścia nieużywane. Tak więc zapiszemy:

```
MOV A, #11111000b
MOV DPTR, #IOCR
MOV @DPTR, A
```

Jako ostatni zostaje rejestr PAMR. Ustawmy adresowanie portów PA...PF jako bitowe, co pozwoli na oddzielne sterowanie każdym pinem układu. Dodatkowo, w przypadku, kiedy wyjście przerwania INT układu SA9203 połączone będzie z jednym z wejść procesora INT0 lub INT1 (poprzez jumper JP1) dobrze jest profilaktycznie ustawić polaryzację sygnału zgłoszenia przerwania, mimo iż w naszym przykładzie ta funkcja nie będzie wykorzystywana. Można to zrobić ustawiając bit 0 w rejestrze PAMR. Bit 1 rejestru powinien być wyzerowany – funkcja przerwania jest nieaktywna.

Instrukcja konfigurująca rejestr będzie następująca:

```
MOV A, #11111101b
MOV DPTR, #PAMR
MOV @DPTR, A
```

I to już wszystko. Teraz aby np. ustawić poziom wysoki na końcówce 3 portu PC należy wykonać instrukcję:

```
MOV DPTR, #BITPC3 ;adres pinu PC3
MOV A, #1 ;wpisanie jedynki do bitu portu
MOV @DPTR, A
```

Podobnie można postąpić z każdym innym portem.

W drugim przykładzie wykorzystamy generowanie sygnału przerwania przez układ SA9203 przy odbiorze danych z portu PA, kiedy to zewnętrzne urządzenie chce przesłać daną do tego portu. Sytuację tę ilustruje rys.11

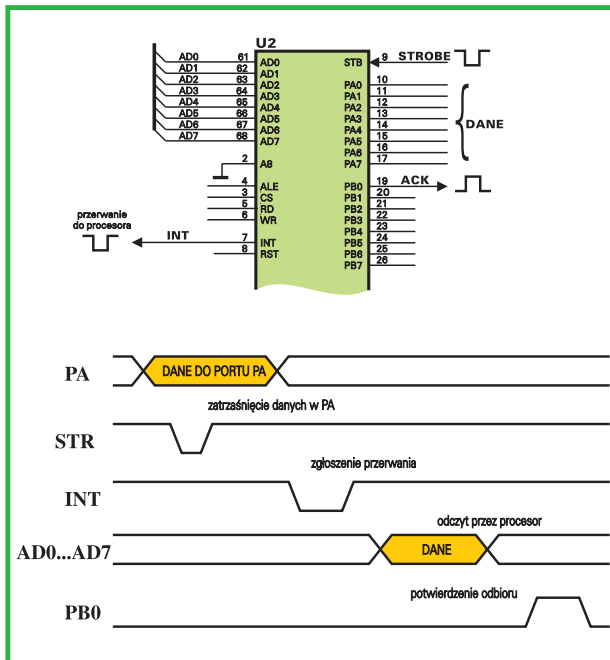
Sytuacja na rysunku przypomina przykład przedstawiony w poprzednim odcinku klasy mikroprocesorowej, kiedy to omawiałem układ 8255 i jego współpracę z drukarką. Wracamy jednak do naszego przykładu. Do prawidłowej transmisji danych z potwierdzeniem wykorzystamy:

- port PA do transmisji danych
- linię STB do potwierdzania zapisu danych do PA przez urządzenie zewnętrzne
- linię PB0 do potwierdzenia odbioru danych przez procesor z SA9203
- linię INT układu SA9203 do zgłoszenia przerwania

Przjrzyjmy się przebiegom z rys.11. Urządzenie zewnętrzne wystawia na linię portu PA daną. Potwierdza to przez podanie stanu niskiego na linię STROBE układu SA9203. Układ ten zapamiętuje daną w wewnętrznym rejestrze PA. Następnie SA9203 zgłasza przerwanie do pro-



## Też to potrafisz



Rys.11 Transmisja danych z urządzenia zewnętrznego do SA9203 z potwierdzeniem.

cesor wystawiając poziom niski na linię INT. Procesor w procedurze obsługi przerwania odczytuje daną z portu PA adresując układ SA9203 – w sposób pokazany wcześniej. Po odbiorze danej procesor powinien poinformować urządzenie zewnętrzne – nadajnik o gotowości na odbiór kolejnej danej. W tym celu korzystając z linii portu PB0 procesor wystawiając nań stan wysoki potwierdza odbiór (ACK) znaku i gotowość na następny.

Zaprogramujemy więc układ do pracy w tym trybie.

1. Najpierw konfigurujemy rejestr PACR:

```
CLR A ;PA jako wejścia
MOV DPTR, #PACR
MOVX @DPTR, A
```

2. Teraz rejestr PAICR:

```
MOV A, #0FFh ;dana zatraskiwana sygnałem STB
MOV DPTR, #PAICR
MOVX @DPTR, A
```

3. Następnie rejestr IOCR:

```
MOV A, #10000010b ;port PB – wyj., PC...PF – wej.
;polaryzacja STB ujemna
;rezystory podciągające PA nieaktywne
```

```
MOV DPTR, #IOCR
MOVX @DPTR, A
```

4. Pozostał jeszcze rejestr PAMR

```
MOV A, #10000011b ;port PB adresowany bitowo
;port PA adresowany bajtowo
;zezwole nie na INT
;polaryzacja INT ujemna
```

```
MOV DPTR, #PAMR
MOVX @DPTR, A
```

I gotowe, układ SA9203 jest gotowy do odbioru danych z potwierdzeniem. Pozostaje jeszcze tylko ustawić odpowiednio procesor tak aby, uaktywnić wybrane jumperem JP1 (rys.10) przerwanie INT0/1. Poniżej przedstawiam przykład kiedy do zgłoszenia przerwania wykorzystano linię INT1. W listingu dodatkowo będziemy wykorzystywać flagę F0 rejestru PSW procesora do detekcji nadejścia znaku w przerwaniu. Bit F0 będzie po prostu ustawiany przez procedurę obsługi przerwania jeżeli zostanie ono zgłoszone. A zerowany po odbiorze znaku. Przed sekwencją inicjującą zapiszmy więc procedurę obsługi przerwania INT1:

```
INT1_proc:
SETB F0 ;ustawienie znacznika o zgłoszeniu
POP DPL
POP DPH
```

POP Acc ;odtworzenie rejestrów DPTR i Acc Bios'a  
RETI ;patrz odcinek klasy o przerwaniach  
Sekwencja inicjująca procesor powinna mieć postać:

```
init_proc:
MOV vectors, #80h ;ustawienie offsetu tabeli
;przerwań w ext. RAM
SETB EX1 ;zezwole nie na INT1
SETB EA ;odblokowanie przerwania
CLR F0 ;wyzerowanie znacznika zgłoszenia
MOV DPTR, #BITPB0
CLR A
MOVX @DPTR, A;ustawienie pinu PB0 w stan niski
RET
```

Następnie możemy zdefiniować procedurę, która czeka na znak a po odbiorze ładuje go do akumulatora:

```
odbierz_znak:
jnb F0, odbierz_znak ;czekanie na pojawienie się
znaku
MOV DPTR, #PORTA
MOVX A, @DPTR ;odczyt znaku z portu PA SA9203
PUSH A ;przechowanie znaku na stosie
MOV DPTR, #BITPB0
MOV A, #1
MOVX @DPTR, A ;ACK = 1, potwierdzenie odbioru
NOP
NOP ;instrukcje NOP w celu opóźnienia
NOP ;sygnału ACK = 0, ilość zależna od
NOP ;tego jaką długość ma mieć ACK, aby być
NOP ;prawidłowo odczytanym przez nadajnik
CLR A
MOVX @DPTR, A ; ACK = 0, zakończenie odbioru
POP A ;odtworzenie akumulatora
CLR F0 ;i jeszcze wyzerowanie znacznika
RET ;zakończenie procedury i powrót
```

Teraz tak stworzone procedury można użyć w programie:

```
CPU '8052.def'
include 'const.inc'
include 'bios.inc'

ORG 8000h
LJMP START
ORG 8013h

INT1_proc:
SETB F0 ;ustawienie znacznika o zgłoszeniu
POP DPL
POP DPH
POP Acc ;odtworzenie rejestrów DPTR i Acc Bios'a
RETI

init_proc:
;tu wstawić ciało procedury j/w

RET

odbierz_znak:
;tu wstawić procedurę j/w

.....
RET

START:
LCALL init_proc ;inicjacja portów i układu przerwania
LCALL odbierz_znak ;wywołanie procedury odbioru znaku
;dalsze instrukcje działające na odebrany znak
.....
END
```

Wnikliwy Czytelnik z pewnością zauważy, że procedurę odbioru znaku z portu SA9203 można umieścić w ciele procedury obsługi przerwania, tak, aby po zgłoszeniu przerwania nowy znak był odczytany prawidłowo. Wtedy procedura ta może wyglądać następująco:

```
INT1_proc:
MOV DPTR, #PORTA
MOVX A, @DPTR ;odczyt znaku z portu
PA SA9203
MOV BUFOR, A ;zapisanie znaku w buforze
MOV DPTR, #BITPB0
MOV A, #1
MOVX @DPTR, A ;ACK = 1, potwierdzenie
odbioru
NOP
```

## Też to potrafisz

```
NOP ;instrukcje NOP w celu
 ;opóźnień
CLR A
MOVX @DPTR, A ; ACK = 0, zakończenie odbioru
SETB F0 ;ustawienie znacznika o
 ;odbiorze znaku
POP DPL
POP DPH
POP Acc ;odtworzenie rejestrów
DPTR i Acc Bios'a
RETI
```

W tym przypadku pojawia się tajemnicza zmienna BUFOR która jest niczym innym jak zdefiniowanym wcześniej adresem w wewnętrznej pamięci RAM procesora np.

```
BUFOREQU 7Fh ;adres komórki przechowującej bajt z SA9203
```

Należy zwrócić jednak uwagę na pewne niedogodności przedstawionej wyżej procedury. Po pierwsze nie jest wskazana umieszczanie zbyt wielu instrukcji NOP ze względu na to że wydłuża to procedurę przetwarzania, i może spowodować zakłócenia w pracy innych przerwań, np. od wyświetlacza, który może zacząć migotać. Po drugie ze względu na to że potwierdzenie odbioru (ACK) jest generowane w procedurze przetwarzania także, może się zdarzyć, że nie zdążymy odebrać znaku z komórki BUFOR, zanim nadejdzie kolejny bajt informacji z urządzenia zewnętrznego, które po potwierdzeniu zapisze nową daną w SA9203. W tym celu warto stworzyć w programie strukturę – tablicę w RAM której zadaniem byłoby przechowywanie danych odbieranych z układu SA9203. Trzeba by jednak dodatkowo użyć także wskaźnika tej tablicy, który określałby miejsce zapisu nowego odebranego znaku i odczyt ostatnio odebranego. Proponuję powyższe zadanie jako pracę domową. Z pewnością temat jest ciekawy a rozwiązań kilka.

Najciekawsze przedstawię w kolejnym odcinku klasy mikroprocesorowej.

W jednym z najbliższych odcinków naszego kursu zapoznam Was z możliwościami jakie odkrywają procesory serii 51 z wbudowaną pamięcią programu, o sposobach ich używania, programowania. Pokażę jak w dość prosty i wygodny sposób, korzystając z kilku dodatkowych urządzeń peryferyjnych, szybko tworzyć programy, testować je a następnie zapisywać w strukturze procesorów np. 87C51, czy 89C51.

Wszystko to z wykorzystaniem i do zastosowania w komputerku edukacyjnym AVT-2250. Tak więc do zobaczenia.

Sławomir Surowiński

## LEKCJA 12

Ostatnim razem pozostawiłem Was z problemem wydrukowania zawartości pamięci ext. RAM z komputerka w postaci jak pokazano poniżej.

Wydruk miał przedstawiać zawartość pamięci o podanym wcześniej z klawiatury komputerka zakresie adresów. Poniżej przedstawiam listing gotowej procedury służącej realizacji tego zamierzenia. (patrz Listing 1)

|         | Adres                                           | Dana Hex | Dana ASCII       |
|---------|-------------------------------------------------|----------|------------------|
| (2600h) | 0E 57 31 C0 50 9A E2 36 7D 0E 9A DF 35 7D 0E 80 |          | .W1.P.6)...5)..  |
| (2610h) | 3E CA 8A 00 74 06 C6 46 FB 01 EB 04 C6 46 FB 02 |          | >...t.F...F..    |
| (2620h) | 8A 46 FB 50 B0 0F 50 9A AA 10 CF 0C 8D 7E FB 16 |          | .F.P.P.....~.    |
| (2630h) | 57 B0 02 50 E8 05 F8 8D 7E FC 16 57 9A 1F 0C CF |          | W.P.....~.W...   |
| (2640h) | 0C 80 3E 7E 89 00 74 07 C6 06 7E 89 00 EB 17 8A |          | ..>~.t.....      |
| (2650h) | 46 FB 3C 01 75 07 C6 06 CA 8A 01 EB 09 3C 02 75 |          | F<.u.....<.u     |
| (2660h) | 05 C6 06 CA 8A 00 89 EC 5D C3 55 89 E5 31 C0 9A |          | .....J.U.1..     |
| (2670h) | 30 05 7D 0E BF 98 89 1E 57 B8 B7 00 50 9A E3 37 |          | 0.).....W...P.7  |
| (2680h) | 7D 0E BF 98 89 1E 57 BF 18 8A 1E 57 9A 96 38 7D |          | .).....W...W.8}  |
| (2690h) | 0E 83 C4 04 BF 98 89 1E 57 9A 5B 38 7D 0E 9A 47 |          | .....W.[8].G     |
| (26A0h) | 0C CF 0C B0 07 50 E8 8F FA 31 C0 9A 16 01 7D 0E |          | .....P.1....J.   |
| (26B0h) | 5D C3 00 55 89 E5 31 C0 9A 30 05 7D 0E 80 3E C8 |          | J..U.1..0.J..>   |
| (26C0h) | 8A 00 B0 00 75 01 40 A2 C8 8A BF 94 89 1E 57 9A |          | .....u. ....W.   |
| (26D0h) | 1F 0C CF 0C C6 06 7B 89 00 C6 06 7E 89 01 C6 06 |          | .....{.....~     |
| (26E0h) | 52 90 FF B0 00 50 BF 32 0C 0E 57 E8 7C FB 5D C3 |          | R....P.2..W.J.]  |
| (26F0h) | 55 89 E5 31 C0 9A 30 05 7D 0E A0 CB 8A 30 E4 48 |          | U..1..0.)....0.H |

Listing 1

```
1 CPU '8052.def'
2
Zbior: "const.inc"
Zbior: "bios.inc"
Zbior: "port8255.inc"
1 ;*****
2 ;Deklaracja adresow portow ukkladu 8255
3 ;*****
4
5 A000 IO_PA equ A000h
6 A001 IO_PB equ A001h
7 A002 IO_PC equ A002h
8 A003 IO_CTRL equ A003h
9
Zbior: "dump.s03"
5
6 ;Zdefiniowane znaki sterujace drukarka
7 000D CR equ 13 ;Carriage Return - znak powrotu glowicy
8 000A LF equ 10 ;Line Feed - znak przesuwu linii na nastepna
9 000C FF equ 12 ;Form Feed - wysuniecie strony z drukarki
10
11 8000 org 8000h
12 8000 028200 ljmp START
13
14
Zbior: "printer.inc"
1 ;*****
2 ;Procedury obslugi drukarki
3 ;*****
4
5 ;*****
6 * PRNACC * Wysyla znak z akumulatora na drukarke
7 ;*****
```

```

8 8003 C083 prnAcc: push DPH
9 8005 C082 push DPL
10 8007 C0E0 push Acc
11 8009 90A000 chkprn: mov DPTR,#IO_PA ;odczyta stanu drukarki
12 800C E0 movx A, DPTR
13 800D 540F anl A,#0Fh
14 800F 6409 xrl A,#1001b ;czy drukarka gotowa ?
15 8011 6017 jz prnok ;tak to drukuj
16 8013 prnerror:
17 8013 120274 lcall CLS
18 8016 757879 mov DL1,#_E
19 8019 757950 mov DL2,#_r
20 801C 757A50 mov DL3,#_r
21 801F 1202C5 lcall CONIN ;czekanie na dowolny klawisz
22 8022 D3 setb C ;ustawienie znacznika błedu (C=1)
23 8023 D0E0 pop Acc
24 8025 D082 pop DPL ;i zakończenie drukowania
25 8027 D083 pop DPH
26 8029 22 ret
27 802A D0E0 prnok: pop Acc ;od tej instrukcji gdy OK !
28 802C 90A001 mov DPTR,#IO_PB
29 802F F0 movx A, DPTR
30 8030 90A000 mov DPTR,#IO_PA ;wysłanie znaku do drukarki
31 8033 E0 wait: movx A, DPTR ;odczyt stanu drukarki
32 8034 5480 anl A,#80h ;czy można wysłać następny znak ?
33 8036 60FB jz wait ;nie to czekaj
34 8038 C3 clr C ;zakoczenie drukowania z info OK !
35 8039 80EA sjmp exit
36
37 ;*****
38 ;* PRNTEXT * Wysyła tekst ASCIIz na drukarkę (dane: mov DPTR,#tekst)
39 ;*****
40 803B RNTXT:
41 803B E0 movx A, DPTR ;pobranie znaku z bufora
42 803C B40001 cjne A,#0,ok1 ;czy znak końca tekstu ?
43 803F 22 ret ;tak to koniec procedury
44 8040 128003 k1: lcall PRNACC ;nie to wydrukuj znak
45 8043 5001 jnc ok2 ;czy błąd drukowania ?
46 8045 22 ret ;tak to zakończ procedurę
47 8046 A3 k2: inc DPTR ;nie to następny znak
48 8047 80F2 sjmp prntxt
49 8049 22 ret
50
51 ;*****
52 ;* PRNINIT * Inicjuje port 8255 na potrzeby drukowania
53 ;*****
54 804A PRNINIT:
55 804A 90A003 mov DPTR,#IO_CTRL
56 804D 7495 mov A,#95h ;inicjacja rejestrów kontrolnych 8255
57 804F F0 movx DPTR,A
58 8050 7405 mov A,#05h ;inicjacja przerzutnika INTRB 8255
59 8052 F0 movx DPTR,A
60 8053 90A002 mov DPTR,#IO_PC
61 8056 7450 mov A,#01010000b ;SELIN=1, RESET=0 (inicjacja), AUTOFD=1
62 8058 F0 movx DPTR,A
63 8059 7470 mov A,#01110000b ;SELIN=1, RESET=1 (praca), AUTOFD=1
64 805B F0 movx DPTR,A
65 805C 22 ret

Zbiór: "dump.s03"
16
17 ;*****
18 ;* Procedura drukowania zawartości pamięci w hexa PRNDUMP *
19 ;*****
20 805D PRNDUMP:
21 805D 120274 lcall CLS ;wyczyść wyświetlacz
22 8060 75785E mov DL1,#_d ;literka "d" na DL1
23 8063 75F005 mov B,#5 ;parametr procedury GETDPTR
24 8066 1203B9 lcall GETDPTR ;wczytaj początek obszaru do drukowania
25 8069 A983 mov R1,DPH ;i zapamiętaj w R1.R2
26 806B AA82 mov R2,DPL
27 806D 75F005 mov B,#5
28 8070 1203B9 lcall GETDPTR ;wczytaj koniec obszaru
29 8073 AB83 mov R3,DPH ;i zapamiętaj w R3.R4
30 8075 AC82 mov R4,DPL
31
32 8077 740D mov A,#CR
33 8079 128003 lcall PRNACC ;powrót głowicy do lewego marginesu
34 807C 908130 mov DPTR,#nagl
35 807F 12803B lcall PRNTEXT ;wydrukowanie nagłówka
36
37 8082 8983 mov DPH,R1 ;początek obszaru do DPTR
38 8084 8A82 mov DPL,R2
39 8086 astrek: ;kolejny rekord 16 bajtów

```

## Też to potrafisz

```

40 8086 7428 mov A,#('
41 8088 128003 lcall PRNACC
42 808B E583 mov A,DPH
43 808D 120235 lcall HEXASCII
44 8090 128003 lcall PRNACC
45 8093 E5F0 mov A,B
46 8095 128003 lcall PRNACC
47 8098 E582 mov A,DPL
48 809A 120235 lcall HEXASCII
49 809D 128003 lcall PRNACC
50 80A0 E5F0 mov A,B
51 80A2 128003 lcall PRNACC
52 80A5 7468 mov A,#'h'
53 80A7 128003 lcall PRNACC
54 80AA 7429 mov A,#')'
55 80AC 128003 lcall PRNACC
56 80AF 12811D lcall prnspc ;wydrukowanie znaku l
57
58 80B2 7D10 mov R5,#16 ;16 bajtów do wydrukowania
59 80B4 E0 nastb: movx A, DPTR ;w postaci np.:
60 80B5 120235 lcall HEXASCII ;12 45 3A 5B 74 09 BC 5A 4F 1E 12 42 54 76
;DC BA
61 80B8 128003 lcall PRNACC
62 80BB E5F0 mov A,B
63 80BD 128003 lcall PRNACC ;dana w postaci XX
64 80C0 7420 mov A,#20h ;spacja
65 80C2 128003 lcall PRNACC
66 80C5 A3 inc DPTR
67 80C6 E583 mov A,DPH ;sprawdzenie czy
68 80C8 8BF0 mov B,R3 ;aby nie koniec adresu
69 80CA B5F009 cjne A,B,ok3 ;do drukowania
70 80CD E582 mov A,DPL
71 80CF 8CF0 mov B,R4
72 80D1 B5F002 cjne A,B,ok3
73 80D4 8002 sjmp asciz ;to gdy koniec rekordu 16 bajtów
74 80D6 DDDC ok3: djnz R5,nastb ;to gdy nie - skok na następny bajt
75
76 80D8 12811D asciz: lcall prnspc
77 80DB 8983 mov DPH,R1 ; ..- * ...%....3.a.
78 80DD 8A82 mov DPL,R2
79 80DF 7D10 mov R5,#16 ;16 bajtów do wydrukowania
80 80E1 E0 nastc: movx A, DPTR
81 80E2 24E0 add A,#0E0h ;sprawdzenie czy znak jest drukowalny
82 80E4 4004 jc znakok ;tzn. czy jego kod jest > 13
83 80E6 742E mov A,#.' ;jeżeli nie to drukuj kropkę zamiast znaku
84 80E8 8001 sjmp druk
85 80EA E0 znakov: movx A, DPTR
86 80EB 128003 druk: lcall PRNACC ;jeżeli tak to drukuj znak

```



## Też to potrafisz

```

87 80EE A3 inc DPTR
88 80EF E583 mov A,DPH
89 80F1 8BF0 mov B,R3
90 80F3 B5F009 cjne A,B,ok4 ;sprawdzenie czy aby nie koniec adresu
91 80F6 E582 mov A,DPL
92 80F8 8CF0 mov B,R4
93 80FA B5F002 cjne A,B,ok4
94 80FD 8013 sjmp finix
95 80FF DDE0 djnz R5,nastc
96 8101 740D mov A,#CR
97 8103 128003 lcall PRNACC
98 8106 740A mov A,#LF
99 8108 128003 lcall PRNACC ;wydrukowanie konca linii
100 810B A983 mov R1,DPH
101 810D AA82 mov R2,DPL
102 810F 028086 ljmp nastrek
103 8112 740D mov A,#CR
104 8114 128003 lcall PRNACC
105 8117 740A mov A,#LF
106 8119 128003 lcall PRNACC ;koniec linii
107 811C 22 ret
108 811D C083 prnspc: push DPH
109 811F C082 push DPL
110 8121 90812C mov DPTR,#space
111 8124 12803B lcall PRNTXT
112 8127 D082 pop DPL
113 8129 D083 pop DPH
114 812B 22 ret
115 812C 207C2000 space db 20h,'l',20h,0
116 8130 20416472
 8134 65732020
 8138 20202020
 813C 20202020
 8140 20202020
 8144 20202020
 8148 20204461
 814C 6E652048
 8150 6578 nagl db ' Adres Dane Hex'
117 8152 20202020
 8156 20202020
 815A 20202020
 815E 20202020
 8162 20202020
 8166 20202020
 816A 20202020
 816E 2044616E
 8172 65204153
 8176 4349490D
 817A 0A db ' Dane ASCII',13,10
118 817B 2D2D2D2D
 817F 2D2D2D2D
 8183 2D2D2D2D
 8187 2D2D2D2D
 818B 2D2D2D2D
 818F 2D2D2D2D
 8193 2D2D2D2D
 8197 2D2D2D2D
 819B 2D2D2D2D
 819F 2D2D db '_____',
119 81A1 2D2D2D2D
 81A5 2D2D2D2D
 81A9 2D2D2D2D
 81AD 2D2D2D2D
 81B1 2D2D2D2D
 81B5 2D2D2D2D
 81B9 2D2D2D2D
 81BD 2D2D2D2D
 81C1 2D2D2D2D
 81C5 2D2D2D0D
 81C9 0A00 db '_____',13,10,0
120
121 ,*****
122 8200 org 8200h ;tak dla czytelności
123 ,*****
124 8200 START:
125 8200 12804A lcall PRNINIT ;zainicjuj 8255 na potrzeby drukowania
126 8203 12805D lcall PRNDUMP ;wywołanie procedury
127 8206 80FE stop: sjmp stop ;wcisnij klawisz M(onitor)
128
129 8208 END

```

Kompilacja zakończona pomyślnie !  
Zbiór: "dump.s03", 467 bajt(ów), 0.2 sekund(y).

## Też to potrafisz

Większość komentarzy zawarłem po średniku w listingu programu. Po włączeniu do kompilacji zbioru "PORT8255.INC" zawierającego deklaracje portów układu 8255 rozpoczyna się właściwy kod programu. Deklaracja

```
ORG 8000h
```

powinna być w razie potrzeby zmieniona jeżeli przewidujesz umieszczenie kodu programu w innym obszarze RAM komputerka.

Po tej deklaracji dołączam zbiór PRINTER.INC zawierający procedury zdefiniowane i omówione w poprzednim odcinku klasy mikroprocesorowej czyli

PRNINIT – procedura inicjacji układu 8255 na potrzeby drukowania  
PRNACC – procedura wysłania znaku z akumulatora na drukarkę  
PRNTXT – procedura wysłania na drukarkę tekstu zakończonego znakiem "0" (zero)

Kolejne linie to już procedura drukowania danych z RAM'u w postaci przedstawionej wcześniej – czyli PRNDUMP. Rozpoczyna się ona zapaleniem na DL1 literki "d" w celu zaproszenia do wpisania z klawiatury komputerka adresu początkowego i końcowego obszaru który ma być wydrukowany. Dane te są zapamiętywane w rejestrach R1.R2 i R3.R4 odpowiednio start i koniec obszaru.

Dalej w liniach 34 i 35 drukowany jest nagłówek czyli tekst spod etykiety "nagl":

Potem to już kolejne linie zawierające adres początkowy (linie 40...56) rekordu 16 bajtów danych, następnie 16 bajtów z RAM w postaci HEX oddzielonych spacjami (linie 58...74), wreszcie te same dane wydrukowane zostają w postaci znaków ASCII (linie: 76...95) i zakończone znakiem końca linii CRLF (linie: 96...99). Dodatkowo w liniach 81

i 82 sprawdzany jest kod znaku. Jeżeli kod jest < 14 to znak należy do tzw. znaków sterujących drukarki i nie powinien być drukowany, a zamiast niego wstawiana jest w linii 83 kropka. Reszta znaków o kodach 14...255 jest drukowana normalnie. W tym miejscu jeżeli ktoś chce zawęzić bardziej zbiór drukowanych znaków może w linii 81 wstawić zamiast wartości 0E0h większą np. 20h, wtedy będą drukowane znaki ASCII od spacji w górę.

Reszta programu to po prostu inicjacja drukarki (linia 125) oraz sprawdzenie działania procedury drukowania obszaru (linia 126).

Na kolejne popołudniowe zimowe wieczory proponuję następujące zadania:

Dokonać poprawek w programie przedstawionym w dzisiejszym artykule, a dotyczącym odbioru danych z SA9203 i umieszczania ich w buforze o długości np. 128 bajtów, umieszczonym w wewn. RAM procesora 80C52 (adresy: 80h...FFh). Wykorzystać rejestr R1 jako wskaźnik początku bufora (podpowiedź - @R1)

Zapisać sekwencję instrukcji programującej układ SA9203 w postaci:  
- PA jako wejście z potwierdzeniem  
- PB jako wyjście – adresowany bitowo  
- PC jako wyjście – adresowany bajtowo  
- PD...PF jako wejścia adresowane bitowo z aktywnymi rezystorami podciągającymi

Rozwiązania zadań w kolejnym numerze EdW.

Z okazji zbliżających się świąt, wszystkim entuzjastom techniki mikroprocesorowej autor cyklu składa najserdeczniejsze życzenia dalszych sukcesów, korzyści a przede wszystkim satysfakcji z ujarzmiania tych arcyciekawych elementów elektronicznych, do których z pewnością należy przyszłość współczesnej elektroniki. Do zobaczenia w Nowym Roku 1999!

**Sławomir Surowiński**

| Adres | Dana Hex | Dana ASCII |
|-------|----------|------------|
|-------|----------|------------|