Kontynuujemy cykl artykułów poświęconych programowaniu mikroprocesora 8051. W kilku kolejnych odcinkach przedstawimy sposoby dołączania do jednostki centralnej nieskomplikowanych urządzeń zewnętrznych. Zaczynamy ten cykl artykułów opisem rozwiązywania problemu sprzęgnięcia prostej klawiatury z naszym kontrolerem AVT-222.

Programowanie '51, cd. Dołączanie klawiatury, część 1

Każdy mikrokomputer lub sterownik, niech będzie to najprostsza zabawka, ma jakieś urządzenia przeznaczone do kontaktowania się człowieka z maszyną. Do wydawania poleceń maszynie stosuje się przede wszystkim klawiaturę, jest ona bowiem najprostszym urządzeniem komunikacji z komputerem. Do naszych potrzeb użyjemy jej nieskomplikowanych odmian, zbudowanych najwyżej z kilkunastu klawiszy. Bardziej wydumane układy klawiatur nie wniosą niczego nowego do koncepcji ich programowej obsługi; będzie to tylko bardziej skomplikowana procedura, ale zasady pozostaną te same.

O ile w poprzednim odcinku nie pokazaliśmy ani jednego rysunku, o tyle teraz bez schematów niewiele da się wyjaśnić. Mamy programować '51, a µP nie jest abstrakcją i ma pracować w określonym środowisku. Oferowane przez nas otoczenie tego mikroprocesora jest najprostsze z możliwych, tym samym nie ma ograniczeń na stosowanie rozwiązań takich na jakie mamy ochotę. Ograniczenia narzucone przez prostotę takiego układu da się ominąć, czy to przez wzbogacenie jego struktury układowej, czy to drogą programową.

Zanim jednak zajmiemy się tym, co w podtytule zawarto - jeszcze parę niezbędnych uwag o minimodule. Nie będziemy przypominać schematu elektrycznego - wystarczy sięgnąć do EP 11/93. W razie potrzeby będziemy również odwoływać się do właściwego odcinka „Programowania '51". Złącza J1 i J2 na rysunkach będą powtórzeniem złącz wyjściowych minimodułu 8051, oferowanego jako kit AVT-222. Aby było jasne, do której linii mikroprocesora będziemy się podłączać, wyprowadzenia obu złącz zostały odpowiednio oznakowane. Po lekturze artykułu o tym module, niejeden Czytelnik zapewne zauważył, że numeracja, oznaczenia i ustawienie nóżek na płytce minimodułu odpowiadają ściśle układowi wyprowadzeń samego mikroprocesora w najpopularniejszej, 40-nóżkowej obudowie dwurzędowej DIP, co znakomicie wykorzystuje przyzwyczajenia bardziej za-awansowanych mikroprocesorowców. Oni znają na pamięć tę numerację. Ale ad rem.

Zadaniem programu obsługi klawiatury jest wykonanie kilku podstawowych czynności:

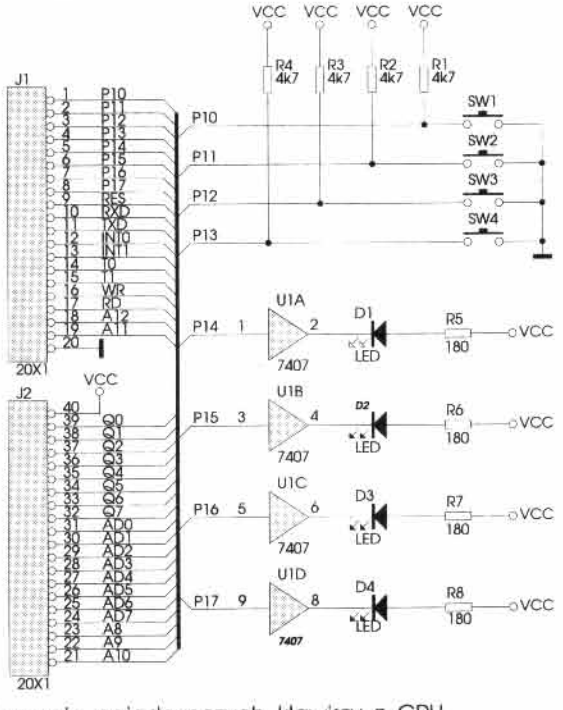
1. Wykrycie naciśniętego klawisza.
2. Likwidacja skutków drgań zestyków klawisza w czasie jego naciskania.
3. Określenie kodu klawisza.
4. Oczekiwanie na zwolnienie klawisza.
5. Likwidacja skutków drgań zestyków klawisza przy jego zwalnianiu.

Procedura obsługi klawiatury może realizować jeszcze więcej zadań, ale te pięć podstawowych musi. Dodatkowe zadania są skutkiem komfortu, jakiego oczekujemy od programu. Zaczniemy zatem od przyłączania do CPU klawiatury kilkuprzyciskowej, a zakończymy na pokazaniu, jak realizuje się struktury matrycowe, te same, za pomocą których ten tekst został przygotowany.

Dołączanie kilku klawiszy

Od liczby klawiszy zależy sposób, w jaki klawiatura zostanie połączona z CPU. Pojedyncze sztuki klawiszy podłączamy bezpośrednio do magistrali danych, tak jak to pokazano na **rys. 1**.

Rezystory R1-R4 służą ustanowieniu stałego, wysokiego stanu czterech młodszych wejść portu P1, gdy wszystkie klawisze są zwolnione. Wprawdzie, w odniesieniu do tego właśnie portu, obecność tych rezytorów nie jest konieczna, bowiem, jak zapewnia producent, port P1 ma zaimplementowaną tego typu rezystancję podciągającą pull-up do napięcia zasilania, zrealizowaną na tranzystorze MOS. Jednak dla porządku nie zaszkodzi te rezystory zostawić, choćby dlatego, że zmieniając port np. na P0, już rezystancji pull-up nie mamy. W takiej konfiguracji układowej każde naciśnięcie klawisza zawsze skutkuje stanem niskim na odpowiednim wejściu.



Rys. 1. Połączenie pojedynczych klawiszy z CPU

Bardzo ważną sprawą są drgania zestyków klawiszy. Klawisz jest zwyczajnym włącznikiem elektromechanicznym, w którym stan jego przewodzenia i nieprzewodzenia uzyskuje się przez mechaniczne zwarcie lub rozwarcie zestyków. Za każdym razem występuje faza przejścia z jednego stanu w drugi. Faza ta objawia się pojawieniem serii mechanicznych drgań, które dają z kolei serię impulsów przewodzenie-nieprzewodzenie. W naszym układzie pojawia się ciąg impulsów zero-jedynkowych. Czas trwania fazy przejściowej zależy od jakości styków klawisza i wynosi ok. 1-10 ms. Z punktu widzenia szybkości działania układów elektronicznych czas 10 ms to niemal wieki. Jeśli założymy, że czas rejestracji zmiany stanu na klawiszu wynosi 1µs, to stosunek obu tych czasów osiągnie 10.000.

Wyobraźmy sobie co się stanie, jeśli zignorujemy omawiane zjawisko. Procedura odczytu klawiatury jest przeważnie używana przez inną procedurę, która jest z kolei procedurą cykliczną, interpretującą w kółko kody naciśniętych klawiszy. Po zarejestrowaniu faktu naciśnięcia klawisza, zostanie wykonana jakaś czynność zanim zdąży nadejść kolejny impuls fazy przejściowej. Kilka czy kilkadziesiąt impulsów spowoduje tyle samo identycznych reakcji procesora, np. napisania na wielocyfrowym wyświetlaczu jednego, ale tego samego znaku albo wielokrotnego zwiększenia pewnej zmiennej. Użytkownik natomiast ma świadomość, że jednokrotnie nacisnął klawisz i oczekuje odpowiedniej reakcji urządzenia. Żeby nie było tak łatwo, ilość wygenerowanych impulsów jest liczbą przypadkową, czyli programowo nie można „wyciąć" nadmiernej ich liczby. Proces „dogadywania się" z kontrolerem staje się więc udręką, jeśli w ogóle jest możliwy.

Po wykryciu faktu naciśnięcia klawisza program winien zatem odczekać wspomniane 10 ms i potem jeszcze raz odczytać stan klawiatury. Podobnie należy postąpić w czasie oczekiwania na zwolnienie klawisza. Tego czasu nie można dowolnie przedłużać, jego ograniczeniem od góry jest minimalny czas reakcji człowieka.

Zadaniem naszej procedury obsługi klawiatury będzie realizacja pięciu wyżej sformułowanych funkcji oraz dodatkowo zapalenie odpowiadającej przyciśniętemu klawiszowi diody LED, wyłącznie w celu wizualizacji skutków użycia klawiatury. Zamieńmy więc nasze postulaty na ciąg rozkazów mikroprocesora.

```
KEYRD:
            ACALL  P1_RD
            JC  KEYRD1
            RET
KEYRD1:
            ACALL  DEL_10MS
KEYRD2:
            CALL  P1_RD
            JNC  KEYRD
            ACALL  KEY_CODE1
KEYRD3:
            ACALL  P1_RD
            JC  KEYRD3
            ACALL  DEL_10MS
            ACALL  P1_RD
            JC  KEYRD3
            ACALL  KEY_CODE2
            RET
KEY_CODE2:
; PODPROGRAM  CZYNNOSCI  PO  ZWOLNIENIU  KLAWISZA
            MOV  P1,#0FH;
            RET

KEY_CODE1:
; POPROGRAM  ZDEFINIOWANIA  I  ZAPAMIETANIA
; KODU  KLAWISZA  DO  RYS.1
; OD  RAZU  JEST  REALIZOWANA  FUNKCJA  DANEGO
; KLAWISZA
            SWAP  A              ; STAN  KLAWISZY  MA
                                 ; ZAPALAC  DIODY  LED,
            ORL  A,#0FH          ; JEDYNKI  NA
                                 ; MLODSZA  TETRADE
            MOV  P1,A            ; ZAPALMY  TE  DIODY,
                                 ; KTORE  TRZEBA
                                 ; ZAPALIC
            RET                  ; NO  I  JUZ
DEL_10MS:
;POPROGRAM  OPOZNIENIA  10  MS
            MOV  R1,#10
DEL_R1_MS:
; POPROGRAM  REALIZACJI  NIEZALEZNEGO
; OPOZNIENIA  PROGRAMOWEGO
; CZAS  TRWANIA  JEST  WIELOKROTNOSCIA  1MS
; W  R1  LICZBA  MILISEKUND
; DLA  R1=0  OPOZNIENIE  WYNIESIE  256  MS
; UZYWANE  ZMIENNE  :  R0,R1
            MOV  R0,#200         ; W  R0  LICZBA
                                 ; POWTORZEN
DEL_R1_1:
            NOP
            NOP
            DJNZ  R0,DEL_R1_1
            DJNZ  R1,DEL_R1_1
            RET

P1_RD:
; PODPROGRAM  CZYTANIA  P1
; W  ACC  WYMASKOWANY  STAN  KLAWIATURY,
; W  C  FAKT  O  NACISNIECIU  DOWOLNEGO  KLAWISZA
; C=1  -  KLAWIATURA  "RUSZONA",
; C=0  -  KLAWIATURA  NIE  DOTKNIETA
            MOV  A,P1            ; CZYTAMY  PORT  P1
            ANL  A,#0FH          ; ALE  POTRZEBUJEMY
                                 ; TYLKO  4  MLODSZE
                                 ; BITY
            CJNE  A,#0FH,P1_RD1  ;
P1_RD1:     ; WSKAZNIK  CARRY  ZDECYDUJE
            ; O  RUSZANIU  KLAWIATURY
            RET
```

Podprogram obsługi klawiatury o nazwie KEYRD korzysta z kilku podprogramów. W ogóle stosowanie podprogramów w projektowaniu systemów mikroprocesorowych miało zawsze przyszłość. Nie ma na świecie mikroprocesora, który nie dawałby takiej sprzętowej możliwości. Nawet mikroprocesory o tak uproszczonej architekturze, jak seria PIC16C5X (patrz EP 9/94) nie posiadające wielu innych sprzętowych udogodnień, mają jednak rozkaz odwołania się do podprogramu i powrotu z niego. Podprogramy zmniejszają pracochłonność programowania, bowiem te same, kilkukrotnie powtarzające się sekwencje rozkazów pisze się jeden raz, a potem daje odnośnik do tego miejsca w procedurze je wykorzystującej. Z tych samych przyczyn program zabiera w konsekwencji mniej pamięci. Program z podprogramami staje się bardziej przejrzysty, bo jest podzielony na funkcjonalne etapy działania, zwane też modułami. Pozwala to na szybką wymianę samych modułów w całości czy ich edycję, łatwiejsze odnajdywanie funkcji zaimplementowanej tylko w jednym module. Pojęcie modułu będzie tu się przeplatać z pojęciem podprogramu, choć wcale nie są to synonimy.

Pierwszy z tych podprogramów to P1_RD, którego zadaniem jest odczytanie portu P1, wymaskowanie młodszych czterech bitów oraz porównanie ich ze stanem klawiatury nieużywanej. Rozkaz

```
CJNE  A,#0FH,P1_RD1
```

tylko z pozoru wydaje się warunkowym skokiem bez sensu, ponieważ po jego wykonaniu sterowanie programu przenosi się w to samo miejsce niezależnie od relacji równości argumentów. Jak się okazuje, jest to jedyna instrukcja porównania dwóch bajtów przez '51. Procesor wykonuje tu ich niejawne odejmowanie, zaś o jego wyniku informuje wpływając na stan wskaźnika przeniesienia CY, zależnie od tego, czy wystąpiła pożyczka. Wiadomo, że największą liczbą, jaką spodziewamy się dostać w akumulatorze jest tutaj 0FH i z taką też akumulator jest porównywany. Odjęcie od 0FH liczby mniejszej niż 0FH, czyli zawsze, gdy klawiatura została „ruszona", ustawi wskaźnik CY. Jest to dość wygodna sytuacja, bowiem po zakończeniu procedury niezależnie od wyniku dostajemy binarną informację o stanie całej klawiatury, nie wnikając na tym etapie, o który klawisz chodzi.

Oczekiwanie na wciśnięcie klawisza polega tu zatem na czytaniu stanu portu P1 i badaniu stanu wskaźnika CY. Zakończenie podprogramu następuje, gdy naciśnięty klawisz nie został zarejestrowany. Po cóż bowiem czekać na coś, co się nie zdarzyło? Ten czas można wykorzystać do innych celów. Oczywiście, nic nie stoi na przeszkodzie, aby cyklicznie odczytywać stan klawiatury, czyli napisać taką początkową sekwencję rozkazów:

```
KEYRD:      ACALL  P1_RD
            JNC  KEYRD1
            SJMP  KEYRD
```

Po fakcie zarejetrowania naciśnięcia klawisza (CY=1) mikroprocesor przechodzi do wykonywania podprogramu opóźnienia 10 ms. Podprogram o nazwie DEL_10MS realizuje to opóźnienie drogą wielokrotnego wykonywania pętli składającej się z rozkazów NOP (No Operation). Podprogram posiada dwa liczniki pętli. Pierwszy z nich, dla pętli wewnętrznej, to R0. Zapewnia on odliczenie czasu 1ms przy założeniu, że częstotliwość zegara wynosi 12MHz. Druga, zewnętrzna pętla zlicza liczbę milisekund, zawartą w rejestrze R1. Wprawdzie można w tę procedurę wkomponować tylko jedną pętlę, ale dwie pętle czynią ją bardziej uniwersalną. Nie bez przyczyny postawiono dodatkową etykietę DEL_R1_MS. Wystarczy przed wywołaniem podprogramu DEL_R1_MS zapamiętać w R1 liczbę milisekund, różną od potrzebnych nam dziesięciu, a następnie wywołać podprogram DEL_R1_MS. Chociaż moduł ten nie daje precyzyjnego opóźnienia, jednak jego dokładność jest do tych celów wystarczająca.

Możemy też zaproponować nieco inną procedurę generacji opóźnień. Jak pamiętamy z poprzedniego odcinka, czasy pomiędzy jednym a drugim przerwaniem od timera sięgają kilkudziesięciu milisekund, mogą więc przydać się do tworzenia znacznie dłuższych opóźnień.

Wprawdzie to samo można uzyskać ostatnią metodą zwiększając liczbę operacji NOP w pętli oraz dodając jeszcze kilka pętli, ale program taki zabierze więcej pamięci programu i zajmie więcej komórek pamięci RAM lub więcej rejestrów roboczych. Poniższa procedura, zapewniająca 50ms opóźnienia, jest krótka i lepiej niż poprzednia wykorzystuje zasoby sprzętowe mikroprocesora.

```
DELAY_50MS:
                MOV  A,TH0
DELAY2:
                CJNE A,TH0,DELAY1
                SJMP DELAY2
DELAY1:
                CJNE A,TH0,DELAY1
                RET
```

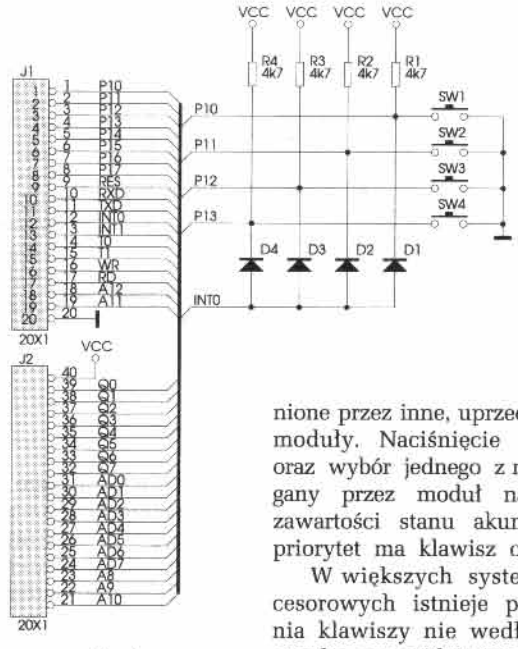
Działanie DELAY_50MS polega na porównaniu stanu rejestru TH0 z sobą samym, lecz w różnych odstępach czasu. Najpierw jedna pętla czeka aż stan TH0 sie zmieni, a potem druga robi coś przeciwnego - oczekuje powrotu stanu rejestru do wartości z początku podprogramu. Oczywiście, podprogram ten wymaga niezatrzymywania timera T0.

Po odczekaniu wspomnianego odstępu czasu następuje ponowny odczyt klawiatury i, jeśli klawisz jest naciśnięty, to stan młodszej tetrady akumulatora zostaje przyjęty jako stan klawiatury. Stan zerowy wskaźnika CY jest teraz interpretowany jako zakłócenie, klawiatura jest nieruszona, czyli sterowanie musi powrócić do początku albo zakończyć procedurę odczytu.

Dopiero po tylu perturbacjach procesor może zająć się właściwą i użyteczną pracą, realizując podprogram dekodujący stan wciśniętych klawiszy. Służy temu podprogram KEY_CODE1. Przesyła on stan odczytanych klawiszy na starszą tetradę portu P1.

Po tym następuje oczekiwanie na zwolnienie klawisza. Tutaj zasady oczekiwania są podobne, co w sytuacji oczekiwania na naciśnięcie klawisza. Wykrycie zwolnienia klawisza jest jeszcze potwierdzone po opóźnieniu i program kończy się zabiegiem, jakim jest wygaszenie diod LED (podprogram KEY_CODE2).

Chociaż KEY_CODE1 i KEY_CODE2 zostały wywołane jeden raz i można było je napisać w bieżącym kodzie modułu KEY_RD, to jednak zapisano je jako osobne podprogramy. Realizują one reakcję procesora, najpierw na zmianę stanu klawisza z wyłączonego na załączony, a potem odwrotnie. Procedury głównej KEY_RD może nie interesować sposób, w jaki klawisze będą interpretowane. Możemy więc zmieniać treść modułów KEY_CODE1 i KEY_CODE2 w zależności od potrzeb. W naszym przykładzie procedura interpretacji klawisza jest jednocześnie ostateczna, czyli po jej realizacji nic więcej się nia zdarzy. Może jednak się okazać, że czas wykonania polecenia wydanego z klawiatury jest zdecydowanie dłuższy od czasu obsługi klawiszy. Mało tego, informacja, która przyszła od użytkownika będzie potrzebna do innych celów, np. jednemu z przerwań. Dlatego nieco innym rozwiązaniem modułu interpretacji klawisza jest zapis kodu klawisza do wskazanej komórki pamięci RAM. Stamtąd inne procedury odczytają ten kod i go zdekodują. Przykładem jest podprogram KEY_CODE3, zapisujący kod naciśniętego klawisza do komórki pamięci RAM, zwanej tutaj buforem klawiatury. Może on z powodzeniem zastąpić moduł KEY_CODE1, zaś nowej treści KEY_CODE2 nie będziemy tu przedstawiać.



Rys. 2. Zmodyfikowany układ połączenia klawiatury z CPU

```
KEYB        EQU  31H  ;  BUFOR  KLAWIATURY
KEY_CODE3:
;  PODPROGRAM  ZAPISU  KODU  KLAWISZA  DO  BUFORA
;  W  ACC  W  PIERWSZEJ  TETRADZIE  KOD  KLAWISZA
;  JESLI  NACISNIETO  DWA  KLAWISZE,  WYZSZY
;  PRIORYTET  MA  O  MNIEJSZYM  NUMERZE

            MOV  KEYB,#0FFH
KEYC31:
            INC  KEYB    ;  KODEM  PIERWSZEGO
                         ;  KLAWISZA  JEST  00H
            RRC  A       ;  DO  CY  STAN
                         ;  NAJMLODSZEGO  BITU
            JC  KEYC31   ;  TO  NIE  TEN  KLAWISZ
            MOV  A,KEYB
            SWAP  A
            CPL  A
            ORL  A,#0FH
            MOV  P1,A
            RET          ;  TO  TEN  KLAWISZ,
                         ;  W  KEYB  JEGO  KOD
```

Jak pokazano wyżej, bardzo wygodnym jest definiowanie kodów klawiszy jako ciągu liczb naturalnego kodu dwójkowego. Taka procedura jest jedną pętlą, w której inkrementuje się (zwiększa o 1) bufor klawiatury. Jej zakończeniem jest wykrycie zera na kolejnym, analizowanym bicie akumulatora. Założeniem poprawnej pracy tego modułu jest oczywiście na wejściu do niego istnienie stanu akumulatora różnego od 0FFH, czyli obecność w nim co najmniej jednego zera. Jest ono zapewnione przez inne, uprzednio wywoływane moduły. Naciśnięcie dwóch klawiszy oraz wybór jednego z nich jest rozstrzygany przez moduł na etapie analizy zawartości stanu akumulatora: .wyższy priorytet ma klawisz o kodzie niższym.

W większych systemach mikroprocesorowych istnieje potrzeba kodowania klawiszy nie według takiej prostej zasady czy widzimisię programisty, ale na wyjściu oczekujemy np. kodu ASCII klawisza. Taka procedura rozpoznająca jest bardziej złożona, przedstawimy ją omawiając klawiatury matrycowe.

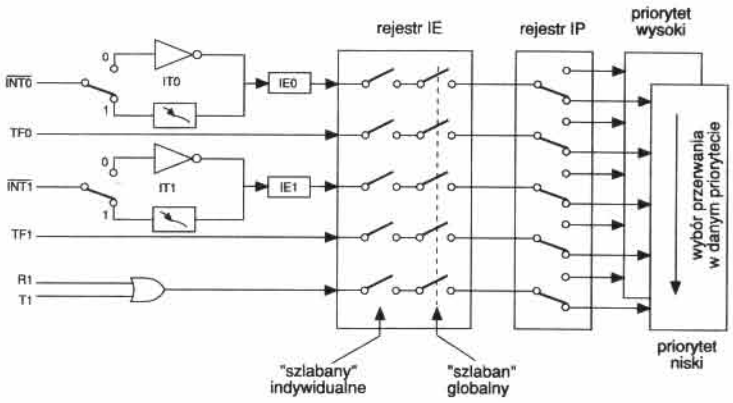
Mikroprocesor, oprócz dekodowania klawiatury ma do wykonania wiele innych, równie lub bardziej odpowiedzialnych czynności. Np. w systemach pracujących w czasie rzeczywistym, gdzie każdy cykl maszynowy jest skrupulatnie liczony i zagospodarowywany na rzecz obsługiwanych procesów, dołączenie klawiatury z rys.1 trzeba uznać za rozrzutność wykorzystania czasu. Procesor bowiem traci czas na bardzo częste sprawdzanie wystąpienia niezmiernie rzadkiego zdarzenia. Zaleca się w związku z tym inne rozwiązanie dołączenia klawiatury - pokazane na **rys. 2**.

Zespół kluczy diodowych D1-D4 razem z wewnętrznym rezystorem pull_up linii INT0 spełnia rolę wielowejściowej bramki NAND. Zadaniem tych kluczy jest stwierdzenie faktu „ruszenia" klawiatury, co jest sygnalizowane niskim stanem na linii INT0.

Nie należy zapomnieć o odblokowaniu przerwania INT0, czyli do modułu inicjującego działanie systemu mikroprocesorowego trzeba dopisać jeszcze dodatkowe linie:

```
SETB  TCON.1
SETB  IE.0
```

Ustawienie bitu TCON.2, który w dokumentacji firmowej jest znany jako IE0 oznacza, że metodą ustawienia przerzutnika przerwania jest pojawienie się opadającego zbocza sygnału na linii INT0. O tym, czy ten fakt będzie wykorzystany do generacji przerwania decyduje następna linijka programu, która jest kolejnym „szlabanem" na drodze sygnału żądania obsługi do systemu przerwań. Ostatnim „szlabanem" jest bit IE.7, globalnie „przepuszczający" wszystkie sygnały, które wcześniej przeszły „szlabany" indywidualne dla każdego przerwa-



Rys. 3. Symboliczny schemat działania "szlabanów"

nia (bity IE.0 - IE.4). Tę zasadę symbolicznie pokazuje **rys. 3**.

Zmiana stanu z wysokiego na niski jest rejestrowana przez przerzutnik IE0,którego ustawienie generuje przerwanie od linii INT0. Napiszmy zatem procedurę takiej obsługi.

```
INTO_SERVIS:
                ACALL  DEL_10MS
                ACALL  P1_RD
                ACALL  KEY_CODE1
                RETI
```

Rozkaz powrotu z podprogramu RETI różni się od rozkazu RET tym, że oprócz odzyskania stanu licznika rozkazów ze stosu dodatkowo system arbitrażu przerwań jest informowany o zakończeniu obsługi przerwania. Jeśli zatem na zakończenie programu będzie tu użyty rozkaz RET, spodziewać się należy, że zostanie zakłócona obsługa przerwań.

Powyższa procedura jest niby w porządku, wypełnia ona wcześniej sformułowane zasady. Jej wadą dalej pozostaje długi czas wykonania (podprogram DEL_10MS), z czym chcemy walczyć. A tutaj staje się to bardzo kłopotliwe. Trzeba wiedzieć, że przerwanie zewnętrzne INT0 ma najwyższy poziom obsługi w danym priorytecie. Oznacza to, że pozostałe przerwania będą nieobsługiwane w czasie wykonywania modułu INT0_SERVIS. Przez ponad 10000 cykli procesor nie będzie robił czegokolwiek pożytecznego, tylko będzie zliczał rozkazy NOP! Przyjmijmy więc przy okazji generalną zasadę, że procedury obsługi przerwań powinny być szybkie. Zmieńmy zatem koncepcję odczytu klawiatury tak, żeby jednak procesor zwolnić z wykonywania bezsensownej pracy.

Pomocną koncepcją może być wielozadaniowość. Niby wielkie słowo, godne dużych procesorów, ale czemu by nie spróbować zrealizować tej koncepcji właśnie tu i teraz, zwłaszcza, iż jej namiastki '51 posiada? Spróbujmy.

Każdy znany nam mikroprocesor działa tak samo: pobiera rozkaz (ang. fetch) a potem go wykonuje (ang. execute). Jeśli nawet następuje przerwanie, to powyższa filozofia działania nie zmienia się. Procesor wykonuje inne, o wyższym priorytecie zadanie. To jest właśnie namiastka wielozadaniowości.

Zwróćmy uwagę na to, że w wielu systemach niektóre procesy nie wymagają ciągłego zaangażowania procesora. W tym czasie procesor może wykonywać inne czynności. Do kierowania ścieżki logicznej programu na właściwy tor służą tzw. semafory. Są to przeważnie jednobitowe znaczniki, takie same, jakich mikroprocesor używa do generowania przerwań. Decydują one, który proces zostanie uruchomiony, z tą różnicą, że w strukturze mikroprocesora są realizowane sprzętowo, zaś nasze będą programowe. Moduł koordynujący bada stan znaczników i na podstawie ustalonych priorytetów i innych relacji podejmuje decyzję, który proces w danej chwili uruchomić. Jednym z takich procesów może być obsługa klawiatury.

Ustalmy więc dwa znaczniki, które będą nam służyć za semafory do inicjacji określonych reakcji procesora na sygnały z klawiatury. Pierwszy z nich (KEY_PRESS) będzie informował o fakcie naciśnięcia jednego z klawiszy, a drugi (AFTER_DEL) o początku generacji opóźnienia. W jaki jednak sposób przeczekać drgania zestyków klawiszy nie angażując do tego programu głównego? Wszak '51 to maszyna synchroniczna, której zatrzymać się tak łatwo nie da. Zresztą nie chcemy czegokolwiek zatrzymywać, mamy życzenie, aby mikroprocesor jednak czymś się „zajął". Tu z pomocą przychodzą zasoby sprzętowe mikroprocesora. Mamy do dypozycji timer, którego w poprzednim odcinku „nauczyliśmy" odliczać czas. Wykorzystajmy go.

Jak Czytelnik sobie przypomina, timer odliczał 50000 cykli maszynowych, co przy częstotliwości zegara 12MHz dawało 50ms. Jest to czas pięciokrotnie dłuższy niż poprzednio proponowany, ale z drugiej strony proszę wskazać człowieka, który potrafi naciskać klawisz szybciej niż 20 razy na sekundę! Takie opóźnienie uznajmy za dobre. Jeśli ktoś z Czytelników uważa ten czas za niedobry, niech zmodyfikuje liczbę ładowaną do pary rejestrów TH0TL0 według zasad podanych w poprzednim odcinku. Procedura obsługi timera ulegnie rozszerzeniu o następujące linie programowe:

```
KEY_PRESS   EQU  00H  ;  STAN  RUSZENIA  KLAWIATURY
AFTER_DEL   EQU  01H  ;  FAKT  OPOZNIENIA  ODCZYTU
                      ;  KLAWIATURY

;  OBSLUGA  PRZERWANIA  T0
T0_SERVIS:  PUSH  ACC
            PUSH  R0REG
            PUSH  R1REG
*****************************
ustalenie  czasu  trwania  miedzy
przerwaniami  (patrz  EP  12/94)
*****************************
T0_S1:
    JNB  KEY_PRESS,T0_S2   ;  BIT  ZEROWANY
                           ;  -  KLAW.  NIE  RUSZONA
    CPL  AFTER_DEL         ;  BIT  ZEROWANY  -
                           ;  KONIEC  OPOZNIENIA
    JB  AFTER_DEL,T0_S2    ;  BIT  USTAWIONY,  CZYLI
                           ;  POCZATEK  OPOZNIENIA
    CLR  KEY_PRESS
    ACALL  P1_RD
    MOV  KEYB,A
T0_S2:
    POP  R1REG
    POP  R0REG
    POP  ACC
    RETI

;  OBSLUGA  PRZERWANIA  INT0
INT0_SERVIS:
    ACALL  P1_RD
    MOV  KEY_PRESS,C
    RETI
```

Właściwy odczyt stanu klawiszy zachodzi w module obsługi przerwania od timera, natomiast procedura obsługi przerwania od INT0 tylko uaktywnia cały proces.

Nieco inaczej sprawa się przedstawia, gdy przerwanie od timera przychodzi znacznie częściej, np. co 1ms. Wtedy tylko warunkowo możemy odczytywać i zmieniać stan semaforów. Warunkiem dopuszczającym do wykonania powyższej procedury może być np. stała liczba przerwań zliczanych w osobnej komórce pamięci. Zapiszmy to.

```
T0_S1:
    JNB  KEY_PRESS,T0_S2
    INC  PART_MSEC
    MOV  A,PART_MSEC       ;  PART_MSEC  -  ZLICZA
                           ;  PRZERWANIA
    CJNE  A,#20,T0_S2      ;  POROWNUJEMY  Z  20
    MOV  PART_MSEC,#0      ;  BEDZIEMY  LICZYC
                           ;  OD  POCZATKU
    CPL  AFTER_DEL         ;  ITD.
    ;.................
    ;.................
```

O programowaniu mikroprocesora współpracującego z klawiaturami matrycowymi opowiemy w następnym odcinku.

Mirosław Lach