

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 wystarczyło 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 zapoznaliśmy 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 przerwania. 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 niezależnie odbiór znaków portu szeregowego (szczególnie wtedy gdy nadchodzą one w nieokreślonych przedziałach czasowych) od wykonywania pętli głównej 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 przerwania. 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 uaktywnić lub blokować generowanie wybranych przerw. Poszczególne bity tego rejestru odpowiadają za generowanie przerwania od określonego podbloku mikrokontrolera, a dodatkowo najstarszy bit tego rejestru pozwala na bezwarunkowe wyłączenie systemu przerw. 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	ACh	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 przerw (=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 przerw
```

W przypadku chęci uaktywnienia kilku przerw np. z licznika T0 i układu transmisji szeregowy UART, można wykonać następujące operacje:

```
SETB    ETO    ;uaktywnienie przerwania od T0
SETB    ES     ; uaktywnienie przerwania od UART a
SETB    EA     ;globalne odblokowanie przerw
```

Ponieważ wszystkie bity maskujące przerwania znajdują się w jednym rejestrze, można w/w instrukcje 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 przerw 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 przerw 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 przerw – o tym za chwilę) ale przez podbloki wykonawcze procesora, zatem może się zdarzyć sytuacja, kiedy to w jednej chwili nadejdą dwa lub więcej przerw – 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 siebie 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 przerw, każde z nich posiada odpowiedni priorytet, i tak w kolejności od najmniejszego priorytetu do największego jest:

ET2, ES, ET1, EX1, ETO, 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 przerw, 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 przerw? Odpowiedź brzmi nie. Otóż istnieje dodatkowy specjalny rejestr tzw. priorytetów przerw o nazwie IP (ang. „Interrupt Priority” – priorytet przerwania). Znajduje się on pod adresem B8h jak się zapewne domyślał w obszarze SFR procesora. Dzięki niemu można zmieniać priorytety poszczególnych przerw 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	BCh	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 przerw na raz, o kolejności wykonywania poszczególnych procedur obsługi przerw 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 zboczne opadające na wejściu /INT1, przy ustawionym bicie EX1 oraz uaktywnionym systemie przerw (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 przerw: 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 przerw:

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: **tablicy wektorów przerwań**.

Tablicy – 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 wpisujemy 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 wykorzystasz?” 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. Założmy 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 przerzucić procedurę intEX0 w miejsce pomiędzy instrukcję 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 „wyłowić” z wkładki z EdW nr 11/97, gdzie znajduje się skrócony opis wszystkich rejestrów SFR procesora. Oto one:

- a) z wejścia /INT0: bit IT0 (adres: 88h) w rejestrze TCON (88h)
- b) z wejścia /INT1: bit IT1 (adres: 8Ah) w rejestrze TCON (88h)
- c) z licznika T0: bit TF0 (adres: 8Dh) w rejestrze TCON (88h)
- d) z licznika T1: bit TF1 (adres: 8Fh) w rejestrze TCON (88h)
- e) 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)
- f) z licznika T2: bit TF2 (adres: CFh) – dla przepelnienia 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.

88h	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 dołączyć generator przebiegu TTL o takiej częstotliwości, to program procesora w praktyce „zwarowałby”, 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 przerwań. 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 komputera edukacyjnego korzystając ze standardowej procedury A2HEX (dla uproszczenia przyjmujemy wypisanie wyniku bez najstarszego bitu wyniku umieszczonego w C).

Aby to wykonać zapewne posłużą się instrukcjami:

```
MOV A, #skladnik1 ;załadowanie składnika (1)
ADD A, #skladnik2 ;i dodanie składnika 2 (2)
MOV B, #1 ;na 1-szej pozycji (3)
LCALL A2HEX ;wypisz zawartość Acc (4)
..... ;i rób cos 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 zostanie wynik, następuje zgłoszenie jakiegoś przerwania (obojętne jakiego) o program automatycznie skacze do odpowiedniego wskaźnika z tablicy wektorów przerwań, a z tamtą prawdopodobnie w inne miejsce 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 składników, 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 przerwania? 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 zdjecia 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 przerwania, 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 przepelniony – 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 przerwania. Przecież każda instrukcja zajmuje procesorowi określoną ilość czasu! W efekcie np. w sytuacji kiedy przerwanie zewnętrzne (lub z przepelnienia licznika) nadchodzi odpowiednio często – czyli w określonych przedziałach czasu, może dojść do sytuacji, kiedy to w trakcie trwania nie zakończonej jeszcze procedury obsługi przerwania znajdzie ponowny warunek zgłoszenia przerwania. W większości takich przypadków procesor po prostu „zwariuje” a cały program albo się zawiesi, albo pójdzie w przyszłowie „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 zachodzący przerwaniami tak zewnętrznymi jak i wewnętrznymi.

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 resecie 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 jakiegś zmiennej w procedurze obsługi przerwania.

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 assemblera 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 sprawdzić je do czystej praktyki, wspartej jednak krótkimi, lecz niezbędnymi informacjami na temat układów czasowo licznikowych.

Z układami czasowo licznikowymi 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	TM0I
	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 T)
0	1	1	16-bitowy licznik złożony z rejestrów THx. TLx
1	0	2	8-bitowy licznik TLx z automatycznym wpisywaniem wartości początkowej po przepelnieniu 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 licznikowymi procesora. Oto on.

bity: 8Fh 8Eh 8Dh 8Ch 8Bh 8Ah 89h 88h

88h	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	TCON
	7	6	5	4	3	2	1	0	

TF1 (bit TCON.7, adres: 8Fh) – znacznik przepelnienia licznika T1, jest sygnałem zgłoszenia przerwania. Ustawiany jest automatycznie – sprzętowo, zerowany także automatycznie przy przyjęciu przerwania.

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 TOMD i TCON można sterować trybami i zachowaniem się liczników T0 jak i T1.

Jak w praktyce wykorzystuje się liczniki? Otóż podam kilka wskazówek i podpowiedzi które wynikają z codziennego „użytkowania” procesorów 8051 oraz różnorodności ich zastosowań. Dla ułatwienia będą operować 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 odmierzeniem czasu.

Jak wykorzystać licznik do generowania opóźnień w systemie lub do zmusić go do odmierzenia 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 milisekund, 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 odmierzania 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 odmierzać np. 1 ms odstępów czasu (od jednego przepełnienia do drugiego przepełnienia ma upłynąć dokładnie 1 ms), to obliczamy wartość początkową licznika w sposób:

$$\text{wartość początkowa} = 65536 - (1 \text{ ms} / 1 \text{ us}) = 65536 - 1000 = 64536 = \text{FC18h (szesnastkowo)}$$

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 odmierzania 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 zauważmy, 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 uzyskać 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 niezależnej 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 przerwania sprzętowych i liczników i przeanalizujemy procedurę realizującą funkcje 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 prosiutki 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 lekcję 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 dziesiątnej 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 displeju komputerka wartości godzin minut i sekund. W prosty sposób także wyświetlimy tzw. „migający dwukropek” w postaci kresiek (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:

```
1 2 – 3 4 – 5 7
```

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 dziesiąta 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.

- 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.
- 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:

$$T_m = 12 / F_{xtal} = 12 / 11059200 = 1,085069444 \text{ } \mu\text{s (mikrosekundy)}$$

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 najrzejdziej 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 korekcji 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: $T1imp=3600$ (dziesiątne) co w efekcie wyznaczy nam z podanej niżej zależności wartość początkowa licznika T1, która spowoduje że przepelnienie licznika nastąpi dokładnie po 1/256 sekundy.

$$TH1.TL1 = T1max - T1imp + 1 = 65535 - 3600 + 1 = 61936 = F1F0h \text{ (hexadec.)}$$

Zatem wartością początkowa 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 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:

- 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);
- cykle potrzebne na skok do tablicy wektorów przerwań – 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 nijako ujarzmić przerwania 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 przerwań 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 przerwań 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 dyskietce 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 wpisujemy do tej komórki (wew. RAM!) liczbę np. 80h, to tabel wektorów przerwań 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 przerwań a nawet zablokowanie wyświetlacza (który pracuje na przerwaniu od licznika T0) i wykorzystanie go do własnych celów, czego na razie stanowczo odradzam.

Podaję 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 przerwań, 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 EdW) 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:

```
oriTL1, #...
```

```
oriTL1, #...
```

```
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, to w praktyce tryb 0 licznika jest najbardziej wygodnym i pewnym, jeżeli chodzi o generowanie opóźnień niezbędnych do odmierzania 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 wariacie 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 wariacie 1, liczba ta przekracza aktualną pojemność licznika – tym razem 8-bitowego TH1, należy zastosować licznik pośredni na zasadach takich jak poprzednio. Założmy 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 :

$$fz / 128 = 28800 / 256 = 225 (= TH1imp)$$

Podsumowując, można powiedzieć, że jeżeli licznik TH1 zliczy za każdym razem 225 impulsów o częstotliwości fz (28800 Hz) to zajmie mu to dokładnie 1/128 sekundy. Jeżeli do tego dodamy – pośredni licznik zliczy te 128 „ułamek 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 powinien 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”.

Założmy 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 dziesiątka 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, miesięcy 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'				68 8051 D0E0	pop	Acc	
2	*****				69 8053 32	reti		
3	;Klasa mikroprocesorowa – LEKCJA 8				70			*****
4	;Program obsługi zegara czasu rzeczywistego				71 8054	START:		
5	;na komputerkach edukacyjnych AVT-2250				72 8054 C28E	clr	TR1	;licznik T1 stop
6	*****				73 8056 53890F	anl	TMOD,#0Fh	;wyczyszczenie
7	;procedura korzysta z przerwania licznika T1 (tryb 0)							;bitów T1
8	;wykorzystywane są 3 komórki wewn.RAM procesora				74 8059 438900	orl	TMOD,#00h	;T1 jako 16-bitowy
9	;zegar liczy: godziny, minuty i sekundy							; (tryb 1)
10	;w trybie 24-godzinnym				75 805C 758D1F	mov	TH1,#Czest	;załadowanie
11	*****							;licznika
12	include 'const.inc'				76 805F 756300	mov	licz128,#0	;wyzerowanie
13	include 'bios.inc'				77 8062 757280	mov	intvec,#80h	;licznika 1/256 sek.
14								;załadowanie MSB
15	;Definicje komórek w wewn.RAM procesora				78 8065 D2AB	setb	ET1	;wektora przerwan
16	;zajmowane przez dane zegara							;odblokowanie
17								;przerwania od T1
18 0060	GODZ equ 60h		;licznik godzin		79 8067 D2BB	setb	PT1	;priorytet na to
19 0061	MIN equ 61h		;licznik minut					;przerwanie
20 0062	SEK equ 62h		;licznik sekund		80			
21					81 8069 120274	lcall	CLS	;wyczyszczenie
22 0063	licz128 equ 63h		;licznik 1/128 sek					;displeja
23					82 806C 757840	mov	DL1,#_minus	
24					83 806F 757940	mov	DL2,#_minus	
25					84 8072 75F001	mov	B,#1	
26 001F	Czest equ 31		;wartosc pocz.		85 8075 1203A7	lcall	GETACC	;pobranie
			;licznika TH1					;początkowej
27								;godziny
28	*****				86 8078 F560	mov	GODZ,A	
29	;Początek kodu programu				87			
30 8000	org 8000h				88 807A 757B40	mov	DL4,#_minus	
31 8000 028054	ljmp START		;petla glowna od		89 807D 757C40	mov	DL5,#_minus	
			;etyk. START		90 8080 75F004	mov	B,#4	
					91 8083 1203A7	lcall	GETACC	;pobranie
32	*****							;początkowej
33								;minuty
34	;Wektor przerwania od licznika T1				92 8086 F561	mov	MIN,A	
35 801B	org 801Bh				93			
36 801B	intT1: ;początek proc.przer.T1				94 8088 757E40	mov	DL7,#_minus	
37 801B 758D1F	mov TH1,#Czest		;przeładowanie		95 808B 757F40	mov	DL8,#_minus	
			;licznika T1		96 808E 75F007	mov	B,#7	
38 801E					97 8091 1203A7	lcall	GETACC	;pobranie
39 801E 0563	inc licz128		;zwiększenie					;początkowej
			;licznika 1/128sek.					;sekundy
40 8020 E563	mov A,licz128				98 8094 F562	mov	SEK,A	
41 8022 C2E7	clr Acc.7				99 8096			
42 8024 7027	jnz koniecT1				100 8096 757A40	mov	DL3,#_minus	;zapalenie kresek
43								;w postaci
44 8026 E562	mov A,SEK		;zwiększenie		101 8099 757D40	mov	DL6,#_minus	;GG-MM-SS
45 8028 2401	add A,#1		;licznika sekund					; (godzina
			;z korekcja					;wprowadzona !)
46 802A D4	da A		;dziesiątna		102 809C 74FA	mov	A,#250	
47 802B F562	mov SEK,A				103 809E 120295	lcall	DELAY	;odczekanie
48 802D B4601D	cjne A,#60h,koniecT1		;czy SEK > 59?,					;ok, 0,5 sekundy
			;nie to skocz		104 80A1 1202C5	lcall	CONIN	;czekanie na start
49 8030 756200	mov SEK,#0		;tak to wyzeruj					;zegara (klawisz)
			;sekundy		105 80A4 D28E	setb	TR1	;start licznika
			;i koryguj minuty					; (zegara)
50 8033					106			
51 8033 E561	mov A,MIN		;zwiększenie		107 80A6	pokaz:		
52 8035 2401	add A,#1		;licznika minut		108 80A6 E563	mov	A,licz128	
			;z korekcja		109 80A8 30E608	jnb	Acc.6,pejne	;co 1/2 sekundy
			;dziesiątna					;pokazuj na zmianie
54 8038 F561	mov MIN,A				110 80AB 757A00	mov	DL3,#0	;puste DL3 i DL6
55 803A B46010	cjne A,#60h,koniecT1		;czy MIN > 59?,		111 80AE 757D00	mov	DL6,#0	
			;nie to skocz		112 80B1 8006	sjmp	czas	
56 803D 756100	mov MIN,#0		;tak to zeruj minuty		113 80B3 757A40	pejne:	DL3,#_minus	;i kreski na DL3
57 8040			;i koryguj godziny					; DL6
58 8040 E560	mov A,GODZ				114 80B6 757D40	mov	DL6,#_minus	
59 8042 2401	add A,#1		;zwiększenie		115 80B9	czas:		
			;licznika minut		116 80B9 E560	mov	A,GODZ	
60 8044 D4	da A		;z korekcja		117 80BB 75F001	mov	B,#1	;na DL1.DL2
			;dziesiątna		118 80BE 12024E	lcall	A2HEX	;wypisz godziny
61 8045 F560	mov GODZ,A				119 80C1 E561	mov	A,MIN	
62 8047 B42403	cjne A,#24h,koniecT1		;czy GODZ > 23 ?,		120 80C3 75F004	mov	B,#4	;na DL4.DL5
			;nie to skocz		121 80C6 12024E	lcall	A2HEX	;wypisz minuty
63 804A 756000	mov GODZ,#0		;tak to zeruj		122 80C9 E562	mov	A,SEK	
			;godziny		123 80CB 75F007	mov	B,#7	;na DL6.DL7
64					124 80CE 12024E	lcall	A2HEX	;wypisz sekundy
65 804D	koniecT1:				125 80D1 80D3	sjmp	pokaz	;i od początku
66 804D D082	pop DPL		;odtworzenie		126			
			;rejestrów		127 80D3 END			
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