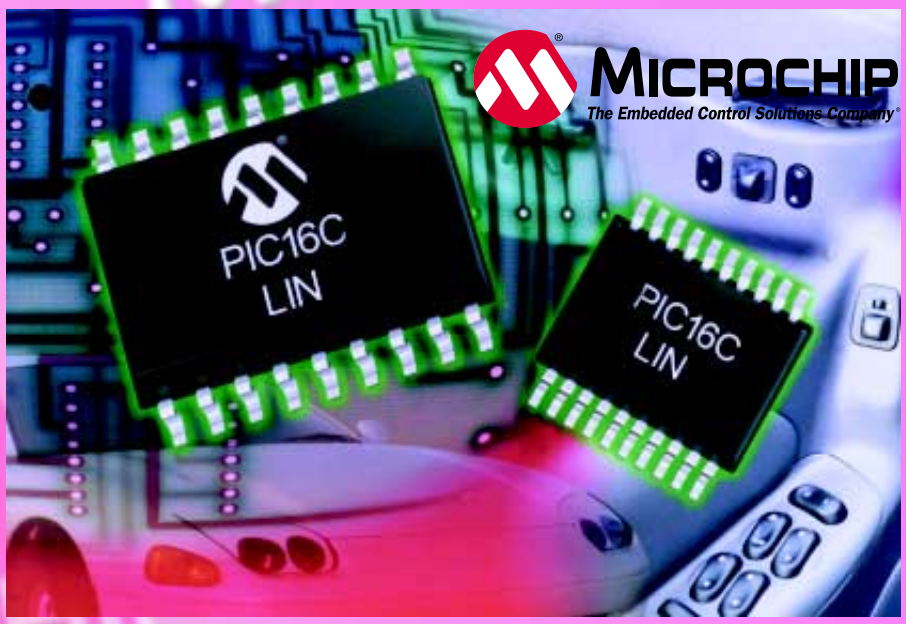
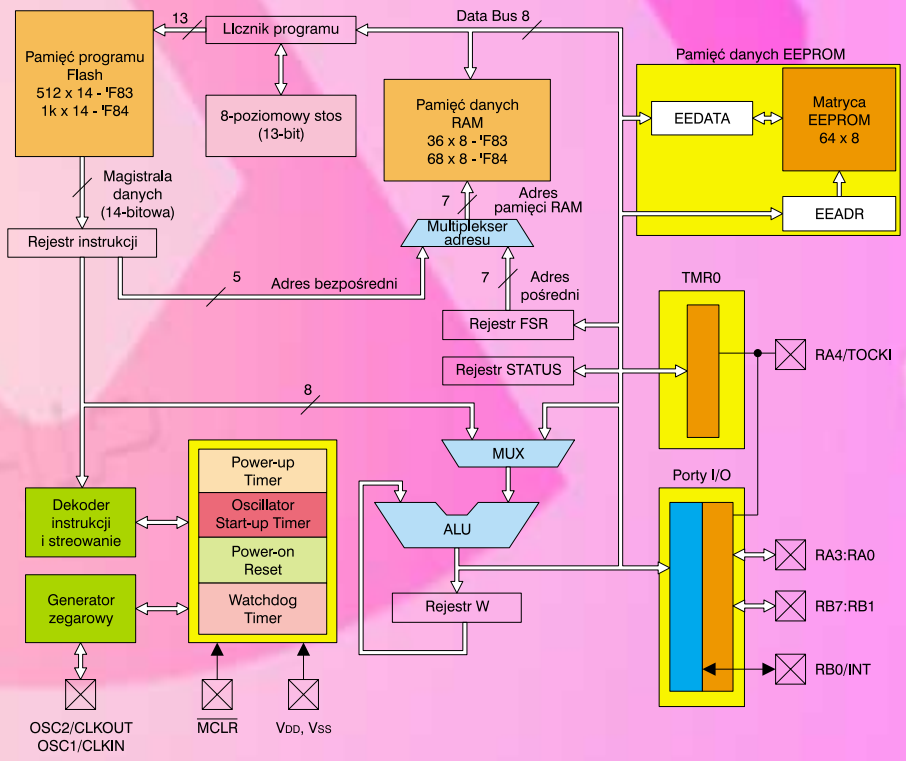
Architektura mikrokontrolerów PIC16F8x

Na życzenie wielu Czytelników, rozpoczynamy cykl artykułów, w których krok po kroku przedstawimy architekturę mikrokontrolerów PIC16F8x firmy Microchip, jednych z najpopularniejszych w naszym kraju.



Mikrokontrolery z intelowskiej rodziny MCS51 dominują w systemach mikroprocesorowych nie tylko w naszym kraju. O tym chyba nie trzeba nikogo przekonywać. Dlaczego tak jest? Związane jest to niewątpliwie z bardzo dobrze przemyślaną architekturą i listą rozkazów. Ponadto, lata obecności na rynku zaowocowały wieloma aplikacjami i doskonałymi narzędziami projektowymi. Dodajmy do tego wielu producentów oferujących rdzeń zgodny z MCS51, z wbudowanymi wieloma peryferiami umożliwiającymi współpracę z zewnętrznymi układami peryferyjnymi (np. portem równoległym 8255, portem szeregowym 8251 itp.).

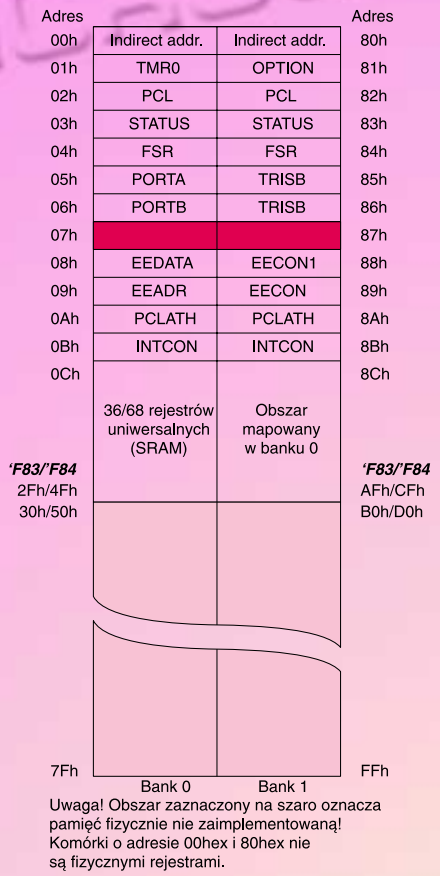
Chociaż „staruszek" ma się dobrze, to chyba nie należy się spodziewać, że taki stan rzeczy będzie trwał wiecznie. Powstają mikrokontrolery, które skutecznie konkurują z tym potentatem. Mają one do zaoferowania wiele nowoczesnych rozwiązań zwiększających funkcjonalność i szybkość działania przy coraz mniejszym poborze mocy.



Rys. 1. Schemat blokowy mikrokontrolerów PIC16F8x.

Jednym z takich opracowań, zdobywających sobie coraz większą popularność w świecie, są mikrokontrolery rodziny PIC16 firmy Microchip. W jej skład wchodzi ponad 60 mikrokontrolerów różnych typów różniących się między sobą rodzajem pamięci programu (Flash, EEPROM, EPROM), liczbą linii portów oraz wbudowanymi peryferiami. Te najbardziej rozbudowane mają wbudowane interfejsy USART, I^2C i SPI, 368 bajtów pamięci RAM, oraz 33 linie portów, które mogą być obciążane prądem 20mA. Wiele z mikrokontrolerów ma wbudowaną pamięć danych typu EEPROM. W artykule skupimy się na architekturze mikrokontrolerów PIC16F8x.

Są to układy, które zdobyły sobie zasłużoną popularność w konstrukcjach amatorskich. Przyczyniło się do tego niewątpliwie stosunkowo łatwe programowanie pamięci programu za pomocą 2-przewodowego interfejsu typu ICSP (ang. In Circuit Serial Programming). Firmowe programatory i systemy emulacyjne mikrokontrolerów firmy Microchip zawsze uchodziły za bardzo dobre, ale niestety drogie. Jednak dla PIC16F83/84 opracowano wiele niekomercyjnych programatorów wraz oprogramowaniem. Ich kompletną dokumentację można znaleźć bez trudu w Internecie. Najprostsze z nich zawierają kilka elementów, nie wymagają oddzielnego zasilacza i wykorzystują złącze szeregowe PC-ta. Jednak programator to nie wszystko. Microchip oferuje darmowy doskonały asembler (w wersji DOS i Windows). Można go ściąg-



Rys. 2. Mapa pamięci danych mikrokontrolerów PIC16F8x.

nąć ze strony *http://www.microchip.com/10/tools/picmicro/code/mpasm/index.htm*. Opublikowaliśmy go także na płycie CD-EP1. Dla bardziej zaawansowanych programistów jest też dostępny linker i bibliotekarz.

Na internetowej stronie producenta dostępnych jest wiele kodów źródłowych dla programowych modułów UART, I²C itp. Opisywany w EP7/00 moduł z PIC16F84 daje też możliwość elastycznego konfigurowania prostych i trochę bardziej skomplikowanych aplikacji mikrokontrolera. Zwalnia to projektanta, przynajmniej na początku, ze żmudnego tworzenia układu prototypowego.

Architektura MCS51 jest opisana w wielu publikacjach w języku polskim. Dużo gorzej jest z mikrokontrolerami PIC. Niniejszy artykuł ma za zadanie tę lukę przynajmniej częściowo wypełnić.

Architektura wielu mikroprocesorów oparta jest na architekturze opracowanej jeszcze w latach 40. przez von Neumanna. W takim rozwiązaniu pamięć programu oraz danych mają wspólną magistralę. W mikrokontrolerach PIC16xx zastosowano architekturę typu Harvard. Pamięć programu i pamięć danych mają własne odrębne magistrale. Odseparowanie obu pamięci pozwoliło na elastyczne dostosowanie długości słowa pamięci programu do przyjętych założeń konstrukcyjnych. W PIC16C8x kody rozkazów mają długość 14-tu bitów i są pobierane w jednym cyklu rozkazowym.

Każdy cykl rozkazowy jest podzielony na dwie fazy: pobranie rozkazu i wykonanie go. Aby zwiększyć wydajność, w czasie kiedy zaczyna się druga faza czyli dekodowanie i wykonywanie rozkazu, jest jednocześnie pobierany następny rozkaz. Tak jest do momentu natrafienia na rozkaz wywołania podprogramu lub rozkaz skoku. Wtedy pobrany następny rozkaz za instrukcją skoku (lub wywołania) jest ignorowany, a cała procedura rozpoczyna się na nowo od adresu, do którego wykonany był skok. Wszystko to powoduje, że przy oscylatorze 10MHz pojedynczy rozkaz jest wykonywany w czasie 400ns. Dość istotną cechą wyróżniającą tę architekturę jest zredukowana do 35 lista rozkazów (RISC). Schemat blokowy mikrokontrolerów rodziny PIC16F8x pokazano na **rys. 1**.

Mikrokontrolery PIC16F8x mają 13-bitowy licznik rozkazów. Może on więc zaadresować do 8kstów pamięci programu. Fizycznie jest zaimplementowanych jednak tylko 1kstów: od adresu 0x0000 do adresu 0x03ff. Po restarcie licznik rozkazów ustawia się na adres 0x0000. Procedury obsługi wszystkich przerwań muszą zaczynać się od adresu 0x0004.

Pamięć danych podzielono na dwa banki (**rys. 2**). W banku 0, od adresu 0x00 do adresu 0x0b (0x0d w 'F84), umieszczono pierwszą część obszaru rejestrów SFR. W obszarze 36/68 bajtów, oznaczonym jako *General Purpose Register* (GPR), jest statyczna pamięć RAM wykorzystywana do przechowywania danych użytkownika. W banku 1, od adresu 0x80 do adresu 0x8b, umieszczono drugą część rejestrów SFR. Adresowanie obszaru od adresu 0x8c do adresu 0xaf powoduje odwołanie się do obszaru GPR z banku 0. Przykładowo, zaadresowanie komórki o adresie 0x0c i 0x8c spowoduje odwołanie się do tej samej danej.

Wybieranie banku realizowane jest poprzez zerowanie lub ustawianie bitu RP0 w rejestrze STATUS np.:

```
bcf STATUS.RP0;wybór banku 0
bsf STATUS.RP0;wybór banku 1
```

Pamięć danych zawiera również 64 bajty pamięci EEPROM. Dostęp do niej możliwy jest tylko poprzez adresowanie pośrednie. Długość słowa we wszystkich komórkach pamięci danych wynosi 8 bitów.

Mikrokontrolery PIC16F8x posiadają 8-poziomowy stos. Jest to zespół 13-bitowych rejestrów, które są umieszczone poza przestrzenią adresową pamięci programu i danych. Zawartości stosu nie można bezpośrednio odczytać ani zapisać do niego własnych danych (brak rozkazów typu *push* i *pop*). Należy pamiętać, że stos jest buforem kołowym co oznacza, że po jego zapełnieniu wysłanie kolejnego (dziewiątego) adresu spowoduje modyfikację zapisanego jako pierwszy.

Rejestry SFR

W rejestrze STATUS REGISTER (**rys. 3**) przechowywane są bity statusu operacji arytmetycznych i logicznych (flagi Z, DC i C) oraz bity wyboru banku pamięci. Jak każdy rejestr, tak i ten, może przechowywać wynik dowolnej operacji. W takim przypadku zapis bitów Z, DC i C jest zablokowany. Bity te są zerowane lub ustawiane tylko poprzez układy logiczne jednostki ALU. Nie jest też możliwe zerowanie lub ustawianie w ten sposób bitów T0 i PD. Widać więc, że efekt działania instrukcji, której wynik jest umieszczony w STATUS REGISTER może być inny od zamierzonego. Do modyfikacji tego rejestru powinny być użyte rozkazy *bcf*, *bsf*, *swapf* i *movwf*, ponieważ nie zmieniają bitów Z, DC i C.

Rejestr OPTION_REG (**rys. 4**) zawiera bity ustalające konfigurację preskalera licznika/timera TMR0 (lub licznika watchdoga WDT), sposób wyzwalania przerwań zewnętrznych oraz źródło impulsów licznika TMR0. Oprócz tego możliwe jest dołączanie wewnętrznego rezystora „podciągania" linii portu POTRB do plusa zasilania.

Wszystkie przerwania (wewnętrzne i zewnętrzne) w mikrokontrolerach PIC16F8x mogą być indywidualnie maskowane. Do tego celu służy rejestr

Tab. 1. Zestawienie podstawowych parametrów mikrokontrolerów tworzących rodzinę PIC16F8x.

Nazwa parametru	PIC16F83	PIC16F84	PIC16CR83	PIC16CR84
Częstotliwość taktowania	10MHz	10MHz	10MHz	10MHz
Pojemność pamięci programu Flash	512	1024	-	-
Pojemność pamięci programu ROM	-	-	512	1024
Pamięć danych RAM	36	68	36	68
Pamięć danych EEPROM	64	64	64	64
Liczba źródeł przerwań	4	4	4	4
Liczba dostępnych linii I/O	13	13	13	13
Napięcie zasilania	2,0..6,0	2,0..6,0	2,0..6,0	2,0..6,0

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

Rys. 3. Rejestr STATUS REGISTER (0x03, 0x83).

R - bit, który można czytać
W - bit, który można ustawiać lub zerować
U - bit nie zaimplementowany
0, 1, x - wartości bitów po włączeniu zasilania (x- wartość nieokreślona)
bit 7: IRP - bit w PIC16F84 nie używany (powinien być wyzerowany)
bit 6...5: RP1...RP0: bity wyboru banku pamięci danych
00 = Bank 0 (0x00...0x7F)
01 = Bank 1 (0x80h...0xFF)
10 = Bank 2 (0x100h...0x17F)
11 = Bank 3 (0x180...0x1FF)
Bit RP1 powinien być wyzerowany. Selekcji banków dokonuje się za pomocą bitu RP0.
bit 4: TO: time-out bit
1 = po włączeniu zasilania, po instrukcji CLRWDT lub instrukcji SLEEP
0 = po przepełnieniu licznika WDT (watchdog)
bit 3: PD: Power-down bit
1 = po włączeniu zasilania lub po instrukcji CLRWDT
0 = przez wykonanie instrukcji SLEEP
bit 2: Z: Zero bit
1 = rezultat operacji arytmetycznej lub logicznej równa się zero
0 = rezultat operacji nie równa się zero
bit 1: DC: bit przeniesienia połówkowego
bit 0: C: bit przeniesienia lub pożyczki dla operacji arytmetycznych wykorzystywany też w rozkazach przesunięcia *rrf* lub *rlf*

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

Rys. 4. Rejestr OPTION_REG (0x81).

bit 7: RBPU: bit sterujący włączaniem podciągania do plusa linii PORTB
1 = PORTB podciąganie wyłączone
0 = PORTB podciąganie włączone
bit 6: INTEDG: rodzaj zbocza wyzwalającego przerwanie zewnętrzne
1 = Przerwanie wyzwalane narastającym zboczem na pinie RB0/INT
0 = Przerwanie wyzwalane opadającym zboczem na pinie RB0/INT
bit 5: T0CS: wybór źródła impulsów licznika/timera TMR0
1 = Impulsy zliczane są z pinu RA4/T0CKI
0 = Źródłem impulsów jest przebieg o częstotliwości oscylatora kwarcowego podzielonej przez 4
bit 4: T0SE: zbocze przy którym następuje inkrementacja licznika/timera TMR0
1 = Inkrementacja następuje przy opadającym zboczu na pinie RA4/T0CKI
0 = Inkrementacja następuje przy narastającym zboczu na pinie RA4/T0CKI
bit 3: PSA: przyporządkowanie preskalera
1 = Preskaler przyporządkowany do WDT
0 = Preskaler przyporządkowany do TMR0
bit 2-0: PS2:PS0 współczynnik podziału preskalera (wartości przedstawiono w tab. 2)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	INTE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

Rys. 5. Rejestr INTCON REGISTER (0x0B, 0x8B).

bit 7: GIE: zezwolenie na wszystkie nie zamaskowane aktualnie przerwania
1 = Przerwania odblokowane
0 = Przerwania zablokowane
bit 6: EEIE: maska przerwania generowanego w momencie zakończenia wpisu do pamięci EEPROM użytkownika
1 = Przerwanie dozwolone
0 = Przerwanie zabronione
bit 5: T0IE: maska przerwania generowanego w momencie przepełnienia licznika/timera TMR0
1 = Przerwanie dozwolone
0 = Przerwanie zabronione
bit 4: INTE: maska przerwania zewnętrznego (zmiana na wejściu RB0/INT)
1 = Przerwania dozwolone
0 = Przerwanie zabronione
bit 3: RBIE: maska przerwania zewnętrznego od zmian na liniach PB4...PB7 portu PORTB
1 = Przerwanie dozwolone
0 = Przerwanie zabronione
bit 2: T0IF: flaga ustawiana w momencie przepełnienia TMR0
1 = przepełnienie TMR0 (musi być zerowana w obsłudze przerwania)
0 = nie ma przepełnienia TMR0
bit 1:flaga INTF: flaga ustawiana w momencie wystąpienia przerwania zewnętrznego RB0/INT
1 = Wystąpiło przerwanie zewnętrzne RB0/INT (musi być zerowana w obsłudze przerwania)
0 = Przerwanie RB0/INT nie wystąpiło
bit 0: RBIF: flaga ustawiana w momencie zmiany na którymś z wejść PB4...PB7
1 = nastąpiła zmiana na liniach PB4...PB7(musi być zerowana w obsłudze przerwania)
0 = na żadnym z tych pinów nie wystąpiła zmiana

INTCON REGISTER, którego budowę pokazano na **rys. 5**.

Z portem A są związane dwa rejestry z obszaru SRF: TRISA (adres 0x85) i PORTA (adres 0x05). Strukturę tych rejestrów pokazano są na **rys. 6**.

Ponieważ port A ma tylko 5 linii, to bity Bit5...Bit7 nie mają swojego fizycznego odpowiednika w liniach portu A i nie są zaimplementowane. Podczas operacji czytania obu rejestrów na tych bitach ustawiane są zera.

Rejestr TRISA służy do określania czy dana linia jest wejściowa, czy wyjściowa. Ustawienie (wpisanie jedynki) odpowiedniego bitu TRISA powoduje ustawienie odpowiadającej temu bitowi linii jako wejściowej (po włączeniu zasilania lub zerowaniu wszystkie linie są wejściowe). Wyzerowanie któregoś z bitów powoduje ustawienie odpowiadającej mu linii jako wyjściowej. W ten sposób ustawiając lub zerując bity rejestru TRISA można dowolnie ustawiać linie portu A jako wejściowe lub wyjściowe.

Linie RA0...RA3 są liniami o poziomach TTL (wejście i wyjście). Linia RA4 ma na wejściu bramkę Schmitta, natomiast wyjście jest typu otwarty dren. Linia ta może też być wejściową dla zewnętrznych impulsów licznika/timera TMR0 (patrz bit T0CS w OPTION_REG - rys. 4).

Odczytując zawartość rejestru PORTA możemy stwierdzić, jakie stany występują na pinach portu A. W momencie zapisu do PORTA jego zawartość jest przepisywana do przerzutników linii portu. Jeżeli teraz odpowiedni bit TRISA jest równy zero, to wartość zapisana do przerzutnika pojawia się na linii portu.

Port B jest również powiązany z dwoma rejestrami SFR: TRISB (adres 0x86) i PORTB (adres 0x06). Ponadto opisany wyżej bit RBPU w rejestrze OPTION_REG steruje „podciąganiem" pinów portu do plusa zasilania (**rys. 7**). Komórki zaznaczone na szaro nie dotyczą portu B.

Ustawienie odpowiedniego bitu TRISB powoduje przełączenie odpowiadającej mu linii w stan wysokiej impedancji. W trybie wejściowym można dołączyć do wszystkich linii rezystor „podciągający" do plusa zasilania (a właściwie tranzystor MOS spełniający funkcję rezystora). Realizuje się to przez wyzerowanie bitu RBPU w rejestrze OPTION_REG (rys. 4).

Wyzerowanie odpowiedniego bitu TRISB powoduje ustawianie odpowiadającej mu linii w trybie wyjściowym. W tym momencie następuje też auto-

Tab. 2. Współczynniki podziału częstotliwości przez preskaler dla liczników TMR0 i WDT.

PS2...PS0	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Adres		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	(1)	(2)
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	---1 1111	---1 1111

Rys. 6. Rejestry TRISA (0x85) i PORTA (0x05).

x - wartość nieokreślona
u - wartość nie zmieniana
(1) wartości po włączeniu zasilania
(2) wartości po zerowaniu

Adres		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	(1)	(2)
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
66h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h	OPTION REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Rys. 7. Rejestry TRISB (0x86) i PORTB (0x06).

(1) wartości po włączeniu zasilania
(2) wartości po zerowaniu
x - wartość nieokreślona
u - wartość nie zmieniana

matyczne wyłączenie podciągania do plusa zasilania dla tej linii.

Zmiany na czterech liniach portu B RB4...RB7 mogą generować przerwania. Oczywiście, zmiany te dotyczą tylko linii ustawionych jako wejściowe. Stany na tych liniach porównywane są z poprzednią wartością wpisaną do specjalnie do tego celu przeznaczonych przerzutników. Wpisywanie do tych przerzutników następuje w momencie odczytu rejestru PORTB. Widać więc, że przerwanie jest generowane w momencie wystąpienia zmiany na liniach od czasu ostatniego czytania PORTB. W obsłudze przerwania należy odczytać PORTB i wyzerować flagę RBIF w INTCON REGISTER. Aby przerwanie mogło być zgłoszone, należy też ustawić jedynkę na bicie maski RBIE w INTCON REGISTER (rys. 5).

Linia RB0/INT spełnia rolę wejścia przerwania zewnętrznego. Bit INTEDG w OPTION REGISTER określa rodzaj zbocza, przy którym to przerwanie zostanie przyjęte (jeżeli bit maski INTE w INTCON REGISTER będzie jedynką). Procedura obsługi przerwania musi zerować bit flagi INTF (INTCON REGISTER). Wszystkie linie portu B są liniami zgodnymi ze standardem TTL.

Linie obu portów mogą być konfigurowane jako wejściowe lub wyjściowe. Poprzez zmianę bitów w rejestrach TRIS można w trakcie wykonywania programu tworzyć linie dwukierunkowe. Zaleca się w takim przypadku bardzo ostrożne wykorzystywanie instrukcji manipulującymi bitami typu *bsf* lub *bcf*. W trakcie ustawiania bitu linii wyjściowej za pomocą takiej instrukcji, wykonywana jest sekwencja:
- odczyt całego rejestru, np. PORTB,
- wykonanie operacji bitowej,
- zapisanie wyniku do PORTB.

W rejestrach wyjściowych PORTB będą teraz wartości chwilowe odczytane z linii portu. W momencie zmiany kierunku innej linii z wejściowej na wyjściową, wystawienie nie kontrolowanych stanów może spowodować przekroczenie dopuszczalnego prądu linii i uszkodzenie układu. Jeżeli linie pozostają w trybie wejściowym ten problem oczywiście nie występuje.

Teraz przedstawimy kilka przykładowych sekwencji inicjacji i obsługi portów. Prezentowana poniżej:

```
bsf STATUS,RP0 ;wybór banku 1
movlw b 00010101
movwf TRISA
;wpisanie wartości do TRISA
```

powoduje, że linie RA0, RA2 i RA4 są ustawione jako wejściowe, natomiast linie RA1 i RA3 jako wyjściowe. Jeżeli linie RA0, RA2 i RA4 pozostaną jako wejściowe to bity portu można ustawiać za pomocą instrukcji:

```
bcf PORTA,1     ;na RA1 wyślij 0
bsf PORTA,3     ;na RA3 wyślij 1
```

Można też wpisywać całą wartość do PORTA:

```
movlw 0xff
movwf PORTA     ; RA1 i RA0 = 1
```

Dobrym, a przede wszystkim bezpiecznym sposobem realizacji bitowych operacji na portach I/O jest zdefiniowanie rejestru (RAM użytkownika), który będzie „cieniem“ rejestru PORTA lub PORTB. Wszystkie operacje ustawiania bitów wykonuje się na tym rejestrze za pomocą np. rozkazów *bcf* i *bsf*, a następnie jego zawartość przepisuje się do rejestru portu:

```
C_PORTB equ 0x0c
;rejestr "cienia"
LED  equ 0x01
;zmienna bitowa odp. PB1
;(zdef. jako wyjście)

;inicjalizacja C_PORTB
;- jesteśmy w banku 0
movf PORTB,w
;do w zawartość PORTB

movwf C_PORTB
;do C_PORTB zawartość w

bcf C_PORTB,LED
;wyzerowanie bitu LED

movf C_PORTB,w
;do w zawartość C_PORTB
movwf PORTB

bsf C_PORTB,LED
;ustawienie na 1 bitu LED

movf C_PORTB
movwf PORTB
```

Linie wejściowe można testować rozkazami testowania bitów:

```
WE0  equ 0
;zmienna bitowa odp. PB0
;(zdef. jako wejście)

movf  PORTA,w
;do W zawartość PORTA
btfsc w,WE0
goto et         ;skok gdy PA0=1
movlw 0xff
;ta instrukcja się wykona,
;gdy PA0=0
```

Tomasz Jabłoński, AVT
tomasz.jablonski@ep.com.pl