Politechnika Łódzka
Instytut Fizyki

Laboratorium elektroniki

Ćwiczenie E58

Wprowadzenie do programowania mikrokontrolerów PIC16 w języku asembler

Wersja 1.0 (29 stycznia 2016)

Spis treści:

Przed zapoznaniem się z instrukcją i przystąpieniem do wykonywania ćwiczenia należy opanować następujący materiał teoretyczny:

1. Podstawy architektury mikrokontrolerów Microchip PIC16 z rodziny Midrange [1, 2, 4-7].
2. Podstawy języka asembler dla mikrokontrolerów Microchip PIC z rodziny Midrange [1-3].
3. Ogólna orientacja w składnikach zestawu ZL4PIC do uruchamiania mikrokontrolerów [8].

1. Cel ćwiczenia

Celem ćwiczenia jest nabycie praktycznych umiejętności programowania mikrokontrolerów PIC16 z rodziny Midrange w języku asembler oraz obsługi programatora mikrokontrolerów i zestawu uruchomieniowego. Programowanie układów peryferyjnych dostępnych w mikrokontrolerach zostało ograniczone w tym ćwiczeniu do dwustanowych uniwersalnych wejść i wyjść cyfrowych. Uruchomienie i testowanie przygotowanego programu odbywa się w zestawie uruchomieniowym ZL4PIC. W ćwiczeniu wykorzystano wyłącznie darmowe, powszechnie dostępne oprogramowanie.

2. Zagrożenia

Rodzaj	Brak	Małe	Średnie	Duże
zagrożenie elektryczne		+		
zagrożenie optyczne	+			
zagrożenie mechaniczne (w tym akustyczne, hałas)	+			
zagrożenie polem elektro-magnetycznym (poza widmem optycznym)	+			
zagrożenie biologiczne	+			
zagrożenie radioaktywne (jonizujące)	+			
zagrożenie chemiczne	+			
zagrożenie termiczne (w tym wybuch i pożar)	+			

Urządzenia wykorzystywane w tym ćwiczeniu zasilane są bezpiecznym napięciem 9V otrzymywanym z zasilaczy podłączonych do sieci 230V.

3. Wprowadzenie teoretyczne

3.1. Przegląd 8 bitowych mikrokontrolerów PIC

Mikrokontrolery PIC są zaprojektowane według zmodyfikowanej architektury Harvard RISC. Architektura Harvard charakteryzuje się rozdzieleniem pamięci programu i pamięci danych, przy czym długość słowa maszynowego jest dobierana niezależnie w obu rodzajach pamięci. W mikrokontrolerach należących do rodzin PIC10, PIC12, PIC16 i PIC18 jednostka arytmetyczno-logiczna i słowo w pamięci danych są zawsze 8-bitowe. Słowo w pamięci programu może mieć rozmiary:

- 12-bitów (Baseline Architecture) – w układach scalonych PIC10 i PIC12 (oraz wybranych PIC16),
- 14-bitów (Midrange and Enhanced Midrange Architecture) – w układach PIC16 (oraz wybranych PIC12),
- 16-bitów (PIC18 Architecture, dawniej High Performance Architecture) – w układach PIC18.

Najliczniej reprezentowane są układy PIC16, które będą podstawą zajęć w Laboratorium Elektroniki. W pracowni dostępne są mikrokontrolery PIC16F84A, PIC16F819 i PIC16F877A. Mikrokontrolery PIC16F84A reprezentują jedną z najprostszych i najstarszych konstrukcji, natomiast mikrokontrolery PIC16F877A wyróżniają się w grupie Midrange szczególnie bogatym wyposażeniem. Mikrokontrolery PIC16F819 reprezentują pośredni zakres oferowanych możliwości.

Tabela 1. Porównanie mikrokontrolerów dostępnych w Laboratorium Elektroniki.

Wyposażenie	Mikrokontroler		
	PIC16F84A	PIC16F819	PIC16F877A
Pamięć programu (słowa 14-bitowe)	1024	2048	8192
Pamięć danych SRAM (bajty)	68	256	368
Pamięć danych EEPROM (bajty)	64	256	256
Wejścia/wyjścia cyfrowe	13	do 16	do 33
Oscylator taktujący zewn./wewn.	tak / –	tak / tak	tak / –
Przetwornik 10 bitowy A/D	–	1	1
Zegary 8/16 bitowe	1 / 0	2 / 1	2 / 1
Komparatory analogowe	–	2	2
Układy CCP i PWM	–	1	1
Interfejsy transmisji szeregowej	–	SPI, I^2C (Slave)	USART (w tym RS232), SPI, I^2C (Master/Slave)
8-bitowy port równoległy	–	–	PSP

Wszystkie mikrokontrolery z grupy Midrange obsługują taki sam zestaw instrukcji. Ponadto dzięki daleko idącym podobieństwom instrukcji ułatwione jest przenoszenie programu na poziomie tekstu źródłowego w języku asembler na inne mikrokontrolery z instrukcjami 12-to oraz 16-to bitowym. Należy jednak pamiętać, że różnice występujące w układach wejścia/wyjścia, dodatkowym wyposażeniu mikrokontrolerów oraz ich mapie pamięci mogą spowodować konieczność modyfikacji programu nawet w przypadku przenoszenia go w ramach grupy Midrange pomiędzy układami o zgodnych wyprowadzeniach (np. przenoszenie z mikrokontrolera PIC16F84A na PIC16F819).

Spośród licznych rejestrów specjalnych dostępnych w przestrzeni adresowej pamięci danych oraz częściowo w przestrzeni pamięci programu, w niniejszej instrukcji zostaną omówione tylko te wybrane, które mają znaczenie w prostych programach wykorzystujących wyłącznie cyfrowe porty wejścia/wyjścia ogólnego przeznaczenia połączone z przyciskami i diodami LED w zestawie uruchomieniowym ZL4PIC. Ponadto założymy, że program nie będzie korzystał z zaawansowanych środków takich jak: system przerwań, przechodzenie w stan uśpienia, sprzętowe zegary, system „Watchdog", rozpoznawanie przyczyny resetu, operacje na nieulotnej pamięci danych EEPROM, program modyfikujący swój kod w pamięci Flash.

Uniwersalne cyfrowe wejścia/wyjścia są pogrupowane w porty A, B, C, … liczące maksymalnie po 8 linii i kontrolowane przez wspólne 8-bitowe rejestry (np. dla portu A są to rejestry PORTA i TRISA). W mikrokontrolerach PIC16F84A i PIC16F819 zaimplementowano tylko porty A i B, natomiast mikrokontroler PIC16F877A posiada także porty C, D i E, które jednak nie będą wykorzystywane w niniejszym ćwiczeniu.

3.2. Środowisko projektowe

Producent mikrokontrolerów PIC udostępnia na stronie www.microchip.com bezpłatne zintegrowane środowisko projektowe MPLAB IDE, które zawiera następujące składniki:

- MPLAB Editor - edytor tekstu umożliwiający wygodną edycję plików źródłowych i nagłówkowych,
- MPASM – makroasembler obsługujący wszystkie rodziny mikrokontrolerów firmy Microchip,
- MPLINK – program łączący moduły tworzone za pomocą asemblera i kompilatorów języka C w jeden plik z danymi dla programatora, emulatora lub symulatora,
- MPLAB Project Manager – menadżer projektów ułatwiający organizację pracy z projektami zawierającymi wiele plików,
- MPLAB SIM – programowy symulator mikrokontrolerów,
- obsługę zewnętrznych urządzeń programatorów i emulatorów sprzętowych, m.in. programator PICSTART Plus dostępny w Laboratorium Elektroniki,
- system pomocy i pliki z dokumentacją.

Dla mikrokontrolerów PIC z rodzin Baseline i Midrange producent nie dostarcza własnego kompilatora języka C, jednakże instalator pakietu MPLAB IDE umożliwia zainstalowanie kompilatorów innych firm.

Pakiet w wersji używanej podczas aktualizacji niniejszej instrukcji można pobrać poprzez wyszukanie "MPLAB IDE V8.66" na stronie www.microchip.com. **UWAGA:** oprócz MPLAB IDE dostępny jest zasadniczo odmienny pakiet MPLAB X IDE, który nie obsługuje programatora PICSTART Plus używanego w tym ćwiczeniu.

3.3. Pamięć konfiguracji w mikrokontrolerach PIC z rodziny Midrange

Pamięć konfiguracji mikrokontrolera znajduje się w przestrzeni adresowej programu pod adresami zaczynającymi się od 2000h. W niektórych komórkach tej pamięci znajdują się numery identyfikacyjne i kod mikrokontrolera. W bieżącym ćwiczeniu istotne znaczenie ma rejestr konfiguracyjny umieszczony pod adresem 2007h. Adres ten leży powyżej przestrzeni adresowej dostępnej z programu wykonywanego wewnątrz MCU (ang. *Microcontroller Unit*). Wartość tego rejestru należy określić przed zaprogramowaniem mikrokontrolera – zostanie ona zapisana razem z kodem programu.

Słowo konfiguracyjne mikrokontrolerów PIC16F84A

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
bit13													bit0

Słowo konfiguracyjne mikrokontrolerów PIC16F877A

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
bit13													bit0

Słowo konfiguracyjne mikrokontrolerów PIC16F819

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
CP	CCPMX	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	MCLRE	FOSC2	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
bit13													bit0

Oznaczenia:

R = bit do odczytu,
P = bit programowalny,
U = bit niezaimplementowany, odczytywany jako 0
-n = wartość gdy urządzenie nie jest zaprogramowane.

Opisy bitów:

bit 13 **CP**: bit zabezpieczenia przed odczytem pamięci programu
1 = odczyt możliwy,
0 = odczyt całej pamięci programu jest zablokowany.

bit 12 **CCPMX**: bit wyboru linii skojarzonej z funkcjami modułu CCP (Capture, Compare, PWM)
1 = CCP używa linii RB2,
0 = CCP używa linii RB3.

bit 11 **DEBUG**: bit włączenia debugera śledzącego pracę MCU w układzie docelowym
1 = debugowanie wyłączone, linie RB6 i RB7 są wejściami/wyjściami ogólnego przeznaczenia,
0 = debugowanie włączone, linie RB6 i RB7 są przeznaczone do współpracy z debugerem.

bit 10-9 **WRT<1:0>**: bity zabezpieczenia przed zapisem do pamięci programu Flash poprzez rejestr EECON
11 = brak zabezpieczenia,
00,01,10 = zabezpieczenia wybranych obszarów przestrzeni adresowej, efekt zależy od modelu MCU.

bit 8 **CPD**: bit zabezpieczenia pamięci danych EEPROM przed odczytem
1 = odczyt możliwy,
0 = odczyt zablokowany.

bit 7 **LVP**: włączenie funkcji programowania niskim napięciem (*Low-Voltage Programming*)
1 = programowanie niskim napięciem jest włączone, linia RB3/PGM jest zarezerwowana do użytku tylko z programatorem,
0 = podwyższone napięcie +12...14V musi być podane przez programator na linię V_{PP} (inne funkcje tej linii to RA5/ $\overline{\text{MCLR}}$), linia RB3/PGM pozostaje wejściem/wyjściem ogólnego przeznaczenia.

bit 6 **BOREN**: bit włączenia resetu po spadku napięcia zasilającego (*Brown-out Reset*)
1 = reset możliwy,
0 = reset zablokowany.

bit 5 **MCLRE**: bit wyboru funkcji linii RA5/ $\overline{\text{MCLR}}$ /V_{PP}
1 = aktywna funkcja $\overline{\text{MCLR}}$ – reset po zwarciu linii do masy,
0 = aktywna funkcja RA5 (cyfrowe wej./wyj.), wejście resetu wewnętrznie podłączone do V_{DD}.

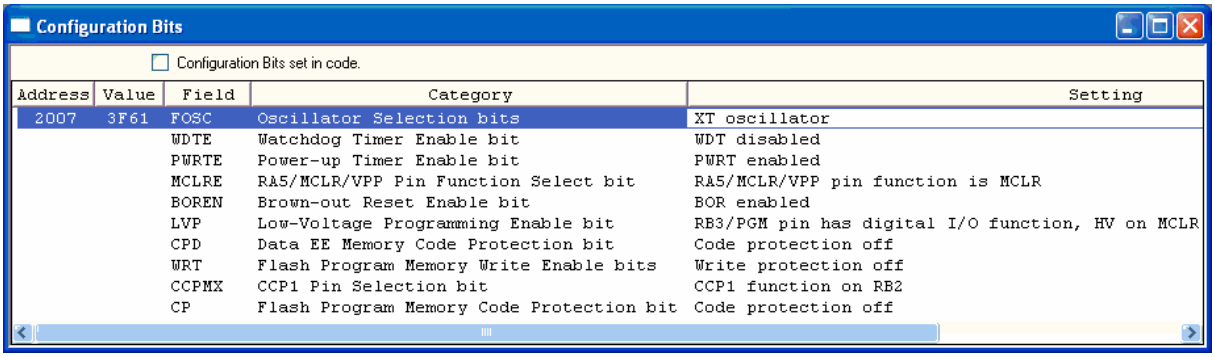
bit 3 $\overline{\textbf{PWRTE}}$: bit włączenia opóźnienia 72 ms dla startu MCU po włączeniu zasilania
1 = opóźnienie wyłączone,
0 = opóźnienie włączone.

bit 2 **WDTE**: bit odblokowania licznika WDT (*Watchdog Timer*)
1 = licznik WDT odblokowany,
0 = licznik WDT zablokowany.

bit 4,1,0 **FOSC<2:0>**: bity wyboru typu oscylatora
111 = zewnętrzny RC z wyjściem zegarowym CLKO na linii RA6/OSC2/CLKO,
110 = zewnętrzny RC, linia RA6/OSC2/CLKO jest portem wej./wyj.,
101 = wewnętrzny RC z wyjściem zegarowym na RA6/OSC2/CLKO; linia RA7/OSC1/CLKI jest portem wej./wyj.,
100 = wewnętrzny RC, obie linie RA6/OSC2/CLKO oraz RA7/OSC1/CLKI są portami wej./wyj.,
011 = w PIC16F819: zewnętrzny sygnał zegarowy podany na RA7/OSC1/CLKI; linia RA6/OSC2/CLKO jest portem wej./wyj.; w PIC16F84A i PIC16F877A: zewnętrzny oscylator RC,
010 = zewnętrzny rezonator kwarcowy w trybie HS; 4-20MHz,
001 = zewnętrzny rezonator kwarcowy w trybie XT; 0,2-4MHz,
000 = zewnętrzny rezonator kwarcowy w trybie LP; 32-200kHz,

Uwaga: w mikrokontrolerach PIC16F84A oraz PIC16F877A bit FOSC2 nie jest zaimplementowany i obowiązują tylko ostatnie cztery kombinacje bitów.

Asembler zintegrowany z pakietem MPLAB posiada komendę „__**config**”, która pozwala na zdefiniowane stanu bitów konfiguracyjnych w tekście źródłowym programu. Alternatywnie konfigurację można ustawić poprzez okno dialogowe Configuration Bits (rys. 1) dostępne po wybraniu z menu Configure pozycji Configuration Bits... . Zestaw kategorii podlegających edycji w tym oknie zależy od aktualnie wybranego typu mikrokontrolera, który można zmienić w oknie dialogowym Select Device (otwieranym z menu Configure, pozycja Select Device...).



Rys. 1. Okno dialogowe Configuration Bits w pakiecie MPLAB IDE v8.66. Przedstawiono ustawienia dla mikrokontrolera PIC16F819 zalecane podczas wykonywania ćwiczenia z niniejszej instrukcji. Ustawienia zalecane dla mikrokontrolerów PIC16F84A oraz PIC16F877A są okrojoną wersją powyższych ustawień.

3.4. Wybór częstotliwości wewnętrznego oscylatora RC w mikrokontrolerach PIC16

Zestaw uruchomieniowy ZL4PIC posiada rezonator kwarcowy 4 MHz przeznaczony do wykorzystania w generatorze taktującym pracę mikrokontrolera. W przypadku użycia mikrokontrolera PIC16F819 dostępny jest także wewnętrzny oscylator RC o częstotliwości 8 MHz, jednakże po włączeniu zasilania częstotliwość ta jest dzielona przez 256. W wielu zastosowaniach ta najniższa możliwa częstotliwość jest niewystarczająca i należy zmienić stopień podziału przez zapisanie odpowiedniej wartości do rejestru OSCCON. Zmiana stopnia podziału może zostać dokonana w dowolnym momencie wykonywania programu. W przypadku użycia oscylatora wykorzystującego zewnętrzny rezonator kwarcowy nie ma możliwości podziału jego częstotliwości. Wybór typu oscylatora może być dokonany tylko na etapie programowania mikrokontrolera (patrz poprzedni rozdział „Pamięć konfiguracji w mikrokontrolerach PIC z rodziny Midrange”)

Pozostałe mikrokontrolery dostępne w Laboratorium Elektroniki (tzn. PIC16F84A oraz PIC16F877A) nie posiadają wewnętrznego oscylatora RC ani rejestru OSCCON.

Rejestr OSCCON, adres 8Fh (*Oscillator Control Register*)

U-0	R/W-0	R/W-0	R/W-0	U-0	R-0	U-0	U-0
—	IRCF2	IRCF1	IRCF0	—	IOFS	—	—
bit 7							bit 0

Oznaczenia:		
R = bit do odczytu,	W = bit do zapisu,	U = bit niezaimplementowany, odczytywany jako 0,
„- n” = wartość po włączeniu zasilania (POR):	„1” = bit ustawiony,	„0” = bit wyzerowany.

Opisy bitów:

bit 7 **Niezaimplementowane**: przy odczycie wartość „0”.

bit 6-4 **IRCF<2:0>**: wybór częstotliwości wewnętrznego oscylatora RC
111 = 8 MHz (bezpośredni sygnał z zegara 8 MHz)
110 = 4 MHz (podzielona częstotliwość z zegara 8 MHz)
101 = 2 MHz (——————— , , ———————)
100 = 1 MHz (——————— , , ———————)
011 = 500 kHz (——————— , , ———————)
010 = 250 kHz (——————— , , ———————)
001 = 125 kHz (——————— , , ———————)
000 = 31,25 kHz (——————— , , ———————)

bit 3 **Niezaimplementowane**: przy odczycie wartość „0”.

bit 2 **IOFS**: bit stabilności częstotliwości wewnętrznego oscylatora
1 = częstotliwość jest stabilna,
0 = częstotliwość nie jest stabilna.

bit 1-0 **Niezaimplementowane**: przy odczycie wartość „0”.

3.5. Odłączanie układów analogowych od zewnętrznych linii mikrokontrolerów

Niektóre mikrokontrolery wyposażone są w dodatkowe układy analogowe. W mikrokontrolerach PIC16F819 i PIC16F877A dostępnych w Laboratorium Elektroniki są to układy analogowych komparatorów oraz przetwornika A/D z wielowejściowym selektorem, natomiast mikrokontroler PIC16F84A nie posiada układów analogowych.

Konfigurację komparatorów analogowych określa rejestr CMCON, natomiast konfigurację przetwornika ADC rejestr ADCON0. Pomimo, iż po włączeniu zasilania wszystkie linie portu A są skonfigurowane jako wejścia analogowe, to jednak układy analogowe są domyślnie wyłączone i nie ma potrzeby zapisywania rejestrów CMCON i ADCON0. Linie skonfigurowane jako wejścia analogowe nie nadają się jednak do wykorzystania jako cyfrowe wejścia/wyjścia. Port A umożliwia skonfigurowanie niektórych linii jako analogowe a pozostałych jako cyfrowe, jednakże w niniejszym ćwiczeniu taka możliwość nie jest użyteczna i założymy, że wszystkie linie zostaną skonfigurowane jako cyfrowe. W tym celu do rejestru ADCON1 należy zapisać wartość 06h (heksadecymalnie).

Rejestr ADCON1, adres 9Fh (*Analog-to-Digital Converter*)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Dotyczy układów PIC16F819 oraz PIC16F877A (rejestr nie występuje w PIC16F84A)

Oznaczenia:		
R = bit do odczytu,	W = bit do zapisu,	U = bit niezaimplementowany, odczytywany jako 0,
„- n” = wartość po włączeniu zasilania (POR):	„1” = bit ustawiony,	„0” = bit wyzerowany.

Opisy bitów:

bit 7 **ADFM**: bit wyboru formatu wyniku przetwornika A/D (analogowo-cyfrowego),
wartość nieistotna w bieżącym ćwiczeniu - opis pominięty.

bit 6 **ADCS2**: bit wyboru zegara taktującego pracę przetwornika A/D,
wartość nieistotna w bieżącym ćwiczeniu - opis pominięty.

bit 5-4 Niezaimplementowane: przy odczycie wartość „0”.

bit 3-0 **PCFG<3:0>**: konfiguracja wejść przetwornika A/D i napięć odniesienia.
W bieżącym ćwiczeniu nie przewiduje się użycia przetwornika A/D. Domyślna wartość 0000 po włączeniu zasilania powoduje skonfigurowanie wszystkich linii portu A jako wejść analogowych. Należy zmienić domyślną wartość na **PCFG = 011x** (gdzie „**x**” oznacza wartość dowolną), która powoduje skonfigurowane wszystkich linii portu A jako uniwersalnych cyfrowych wejść/wyjść. Inne kombinacje są nieistotne w bieżącym ćwiczeniu.

3.6. Wykorzystanie portu A jako cyfrowego portu wejścia/wyjścia

Niektóre linie portu A mogą pełnić różne funkcje związane z dodatkowymi urządzeniami peryferyjnymi. Jeżeli te dodatkowe funkcje są nieaktywne, wówczas linie portu A tworzą dwukierunkowy port cyfrowy zgodny z poziomami napięć układów TTL. Liczba dwustanowych linii dostępnych w porcie A zależy od modelu mikrokontrolera.

Kierunek transmisji danych określają bity w rejestrze TRISA. Ustawienie w tym rejestrze bitu w stan 1 konfiguruje związaną z nim linię portu jako wejście, tzn. sterownik wyjścia linii przyjmuje stan wysokiej impedancji. Po włączeniu zasilania oraz wszelkich rodzajach resetu wszystkie aktywne bity rejestru TRISA ustawiane są w stan 1. Wyzerowanie bitu w TRISA konfiguruje linię jako wyjście, którego stan logiczny jest określony przez odpowiedni bit rejestru PORTA. Niezależnie od wartości zapisanej do rejestru TRISA, bieżący stan linii można zawsze oczytać z rejestru PORTA.

Rejestry kontrolujące stan portu A w mikrokontrolerach PIC16F84A

Adres	Nazwa	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Wartość po POR	po innym resecie
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	---1 1111	---1 1111

Rejestry kontrolujące stan portu A w mikrokontrolerach PIC16F819

Adres	Nazwa	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Wartość po POR	po innym resecie
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxx0 0000	uuu0 0000
85h	TRISA	TRISA7	TRISA6	TRISA5(1)	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	1111 1111

Rejestry kontrolujące stan portu A w mikrokontrolerach PIC16F877A

Adres	Nazwa	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Wartość po POR	po innym resecie
05h	PORTA	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	—	—	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	--11 1111	--11 1111

Oznaczenia:
x = wartość nieznana, u = wartość bez zmiany,
- = bit dotyczący linii, która nie została zaimplementowana w porcie A, odczytywany jako 0,
(1) linia RA5 w PIC16F819 jest tylko wejściem, bit 5 w TRISA jest odczytywany zawsze jako 1.
POR – reset po włączeniu zasilania (*Power On Reset*) oraz po spadku napięcia zasilającego,
inny reset – reset przez wyzerowanie linii $\overline{\text{MCLR}}$ lub przepełnienie licznika WDT (*Watchdog Timer*).

Sterowniki wyjść portu A są typu „push-pull”, tzn. zawierają zarówno tranzystory podnoszące napięcie na wyjściu do poziomu napięcia zasilania jak i tranzystory zwierające wyjście z masą. Wyjścia te można zredukować metodami programowymi do wyjść typu „otwarty dren”. W tym celu należy wyzerować odpowiednie bity w rejestrze PORTA, a następnie przełączać stan wyjść (między wysoką impedancją a zwarciem do masy) przez zapisywanie 1 albo 0 do odpowiednich bitów rejestru TRISA. W mikrokontrolerach PIC16F84A i PIC16F877A (nie dotyczy PIC16F819) sterownik wyjścia linii RA4 jest wyjątkowo typu otwarty dren, tak więc ustawienie 4-go bitu w rejestrze PORTA w stan 1 przełącza wyjście w stan wysokiej impedancji niezależnie od stanu rejestru TRISA.

Uwaga 1: odczyt rejestru PORTA zwraca zawsze bieżący stan linii, nie zaś wartość zapamiętaną podczas ostatniego zapisu do rejestru. Wartość odczytana z PORTA zaraz po zapisie może nie zgadzać się z wartością zapisaną, nawet gdy na wyjściu nie ma zwarcia. Czas przechodzenia wyjścia w zadany stan może być dłuższy od jednego cyklu zegarowego MCU i zależy od pojemności obwodów przyłączonych do wyjścia na zewnątrz mikrokontrolera.

Uwaga 2: operacja zapisu do PORTA przebiega w cyklu odczyt-modyfikacja-zapis. Odczyt bieżącego stanu i zapis stanu po modyfikacji do rejestrów zatrzaskowych dotyczy wszystkich linii portu, nawet podczas wykonywania instrukcji operujących na pojedynczych bitach (tzn. instrukcji bcf i bsf). Jedyną pewną metodą modyfikacji wybranych pojedynczych bitów w PORTA jest wykonanie operacji na buforze w pamięci i zapisanie do rejestru gotowego wyniku instrukcją MOVWF PORTA.

3.7. Wykorzystanie portu B jako cyfrowego portu wejścia/wyjścia

W mikrokontrolerach PIC16F84A, PIC16F819 oraz PIC16F877A port B złożony jest z ośmiu linii. Jeżeli dodatkowe funkcje nie są aktywne, wówczas wszystkie linie portu B można wykorzystać jako uniwersalne wejścia/wyjścia zgodne z poziomami napięć TTL. Praca tych linii jest kontrolowana przez rejestry PORTB oraz TRISB. Sposób użycia rejestrów jest identyczny jak w przypadku analogicznych rejestrów portu A (patrz poprzedni rozdział), z tym że w 8-bitowych rejestrach portu B aktywne są wszystkie bity.

Rejestry kontrolujące stan portu B w mikrokontrolerach PIC16F84A, PIC16F819, PIC16F877A

Adres	Nazwa	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Wartość po POR	po innym resecie
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111

Oznaczenia:
x = wartość nieznana, u = wartość bez zmiany,
POR – reset po włączeniu zasilania (*Power On Reset*) oraz po spadku napięcia zasilającego,
inny reset – reset przez wyzerowanie linii $\overline{\text{MCLR}}$ lub przepełnienie licznika WDT (*Watchdog Timer*).

W mikrokontrolerach PIC16F819 niektóre linie portu B mogą być użyte razem ze specjalistycznymi układami cyfrowymi, takimi jak kontroler magistral szeregowych SPI i I^2C oraz generator PWM (wytwarza impulsy o programowanym wypełnieniu) jednakże układy te nie są aktywne po włączeniu zasilania i wykraczają poza zakres niniejszego ćwiczenia.

Unikalną cechą portu B jest możliwość programowego włączenia rezystorów podciągających napięcie do poziomu napięcia zasilania na liniach skonfigurowanych jako wejścia (1 w odpowiednich bitach rejestru TRISB). Po włączeniu zasilania rezystory są nieaktywne i można je załączyć przez wyzerowanie bitu $\overline{\text{RBPU}}$ w rejestrze OPTION_REG. Bit ten kontroluje załączenie rezystorów na wszystkich wejściach portu B, natomiast na wyjściach (0 w odpowiednich bitach rejestru TRISB) rezystory pozostają odłączone. Rezystory te są konieczne do poprawnego odczytu stanu przycisków podłączonych w zestawie ZL4PIC do wejść mikrokontrolera RB0, RB1, RB2 i RB3. W przypadku gdy rezystory nie są załączone, stan logiczny wejść przy rozwartych przyciskach jest niestabilny.

Uwaga: w pakiecie MPLAB domyślnie ustawione jest niskonapięciowe programowanie MCU (ang. *Low Voltage Program Enabled*), które wymaga użycia dodatkowo linii RB3 i po zakończeniu programowania linia ta nie będzie aktywna jako uniwersalne wejście/wyjście. W zestawie uruchomieniowym ZL4PIC aktywność linii RB3 jest wymagana do odczytu stanu jednego z przycisków. Zalecane jest wybranie programowania podwyższonym napięciem +12V (ang. *HV - High Voltage*), np. w oknie Configuration Bits, Field LVP. Pozostałe linie używane przez programator (RB6 i RB7 z portu B oraz RA5/$\overline{\text{MCLR}}$ z portu A) po zakończeniu programowania są aktywne. Uwaga ta nie dotyczy mikrokontrolera PIC16F84A, który nie obsługuje programowania niskonapięciowego.

Rejestr OPTION_REG, adres 81h

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Dotyczy układów PIC16F84A, PIC16F819 oraz PIC16F877A

Oznaczenia:		
R = bit do odczytu,	W = bit do zapisu,	U = bit niezaimplementowany, odczytywany jako 0,
„- n” = wartość po włączeniu zasilania (POR):	„1” = bit ustawiony,	„0” = bit wyzerowany.

Opisy bitów:

bit 7 **$\overline{\text{RBPU}}$**: bit sterujący rezystorami podciągających napięcie na wyjściach portu B do +5V
1 = podciąganie napięcia w porcie B wyłączone,
0 = podciąganie napięcia w porcie B włączone.

bit 6 **INTEDG**: bit wyboru zbocza wywołującego przerwanie
1 = przerwanie wyzwalane zboczem rosnącym na linii RB0/INT,
0 = przerwanie wyzwalane zboczem opadającym na linii RB0/INT.

bit 5 **T0CS**: wybór źródła sygnału zegarowego dla licznika TMR0
1 = przejścia na linii RA4/TOCKI,
0 = wewnętrzny zegar cyklu instrukcji (CLKO).

bit 4 **T0SE**: bit wyboru zbocza inkrementującego licznik TMR0
1 = inkrementacja po zboczu opadającym na linii RA4/TOCKI,
0 = inkrementacja po zboczu rosnącym na linii RA4/TOCKI.

bit 3 **PSA**: bit powiązania preskalera
1 = preskaler przypisany do WDT (*Watchdog Timer*),
0 = preskaler przypisany do licznka Timer0.

bit 2-0 **PS2<2:0>**: bity wyboru podziału częstotliwości w preskalerze

wartości bitów	podział dla TMR0	podział dla 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

3.8. Struktura programu dla asemblera MPASM

Plik źródłowy dla asemblera MPASM musi być plikiem tekstowym z rozszerzeniem .asm. W pliku tym obowiązują następujące reguły:

- Linia może mieć maksymalną długość 255 znaków.
- Etykiety muszą rozpoczynać się w pierwszej kolumnie. Pierwszym znakiem etykiety musi być litera albo znak podkreślenia „_”. Następnymi znakami mogą być także cyfry. Maksymalna długość etykiety wynosi 32 znaki.
- Rozkazy mikrokontrolera muszą zaczynać się w kolumnie drugiej lub w następnych. Symbol rozkazu musi być oddzielony od pierwszego argumentu co najmniej jedną spacją lub tabulatorem. Kolejne argumenty musza być rozdzielone przecinkiem. Listę rozkazów mikrokontrolerów z rodziny Midrange podano w Aneksie A.
- Komendy asemblera, które nie są bezpośrednimi równoważnikami instrukcji mikrokontrolera, mogą zaczynać się w dowolnej kolumnie. Wybrane komendy asemblera opisano w Aneksie C.
- Komentarz rozpoczyna się znakiem średnika (;). Wszystkie znaki pomiędzy średnikiem i końcem danej linii są ignorowane.
- Liczby zapisane bez wskazania formatu są interpretowane wg formatu ustawionego komendą „list r=” (omówiona dalej). Format liczby można wskazać używając składni przedstawionej w Tabeli 2.

Tabela 2. Formaty liczb i znaków ASCII w asemblerze MPASM.

Format	Oznaczenie	Składnia	Przykład
binarny	Bin	b'*cyfry*'	b'00111001'
ósemkowy	Oct	o'*cyfry*'	o'777'
dziesiętny	Dec	d'*cyfry*' lub .'*cyfry*'	d'100' lub .100
szesnastkowy	Hex	h'*cyfry*' lub 0x*cyfry*	h'9f' lub 0x9f
kod znaku ASCII	–	A'*znak*' lub '*znak*'	A'C' lub 'C'

Typowy niewielki program napisany w języku asembler i składający się tylko z jednego pliku źródłowego zawiera następujące składniki:

1). Deklarację typu mikrokontrolera komendą „list p=" na początku programu.

2). Deklarację domyślnego formatu dla tych liczb, których format nie zostanie jawnie zapisany. W tym celu używa się komendy „list r=", gdzie argumentem po znaku równości może być: „dec" – format dziesiętny, „oct" – format ósemkowy albo „hex" – format szesnastkowy. Komendy „list p=" i „list r=" można połączyć tak, jak pokazano dalej w przykładowym programie.

3). Komendę „#include", która włącza plik nagłówkowy dla wybranego mikrokontrolera do tekstu programu (jeżeli C:\MPLAB jest głównym katalogiem z plikami pakietu MPLAB, to pliki nagłówkowe *.INC znajdują się w katalogu C:\MPLAB\MPASM Suite).

4). Deklarację wartości słowa konfiguracyjnego mikrokontrolera komendą „__config". Argument dla tej komendy można podać bezpośrednio jako liczbę, jednakże bardziej przystępny zapis wykorzystuje stałe zdefiniowane w pliku nagłówkowym, które należy połączyć operatorem bitowego iloczynu logicznego &. Znaczenie poszczególnych bitów słowa konfiguracyjnego zostało omówione w rozdziale „Pamięć konfiguracji w mikrokontrolerach PIC z rodziny Midrange".

5). Definicje stałych symbolizujących adresy zmiennych w pamięci RAM. Mikrokontrolery PIC z rodziny Midrange są zoptymalizowane tylko do bezpośredniego adresowania pamięci (statyczny adres jest zapisany bezpośrednio w kodzie instrukcji). Istnieje także możliwość adresowania pośredniego, jednakże wymaga to dodatkowych instrukcji do obliczania adresów i zapisywania ich do rejestru FSR (adres 04h) przed użyciem do adresowania pamięci. Statyczne adresy zmiennych definiuje się komendą „equ" i zazwyczaj grupuje się po deklaracjach opisanych w punktach 1 ÷ 4 ale przed instrukcjami generującymi kod programu. Przydzielając adresy należy kierować się odpowiednią dla wybranego mikrokontrolera mapą pamięci z Aneksie D i rozważyć taki rozkład zmiennych, aby liczba operacji przełączania banków pamięci została ograniczona do koniecznego minimum. Poniżej zamieszczono przykłady typowych definicji:

```
; zmienne dostępne we wszystkich bankach PIC16F819
Licznik1    equ  0x70
wynik       equ  0x71

; Zmienne w banku 1
timer0_lo   equ  0xA0
timer0_hi   equ  0xA1
```

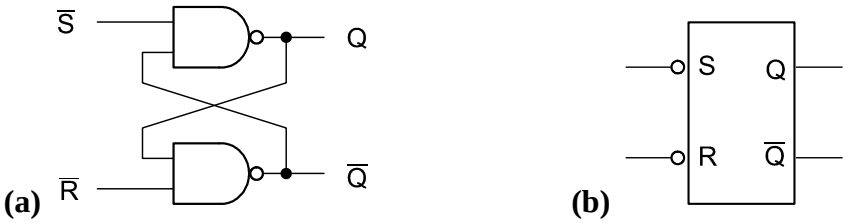
6). Instrukcje generujące kod programu. Przed wejściem w główną pętlę programu zwykle konieczne jest ustawienie stanu początkowego niektórych rejestrów. W mikrokontrolerach PIC z rodziny Midrange przestrzeń danych w pamięci RAM jest podzielona na banki pamięci, z których aktualnie aktywny jest tylko jeden. Oprócz naturalnych instrukcji procesora (zebranych w Aneksie A) charakterystyczne jest częste używanie komend asemblera generujących kod ustawiający aktywny bank, mianowicie „banksel" dla adresowania bezpośredniego (adres w kodzie instrukcji) i „bankisel" dla adresowania pośredniego (przez wskaźnik).

7). Zakończenie pliku źródłowego programu komendą end.

W bardziej rozbudowanych programach przydatne są także inne komendy asemblera omówione w Aneksie C. Opisaną powyżej strukturę ma przykładowy program na następnej

stronie, który symuluje pracę przerzutnika prostego RS przedstawionego na rys. 2. Konfiguracja wejść i wyjść przewidziana dla zestawu uruchomieniowego ZL4PIC jest następująca:

- wejście przerzutnika $\overline{R}$: przycisk S1 przyłączony do wej. mikrokontrolera RB0,
- wejście przerzutnika $\overline{S}$: przycisk S2 przyłączony do wej. mikrokontrolera RB1,
- wyjście Q: dioda LED RA0 przyłączona do wyj. mikrokontrolera o tym samym symbolu,
- wyjście $\overline{Q}$: dioda LED RA1 przyłączona do wyjścia mikrokontrolera o tym samym symbolu.



Rys. 2. Przerzutnik prosty RS wykonany z dwóch bramek NAND: (a) schemat logiczny; (b) symbol.

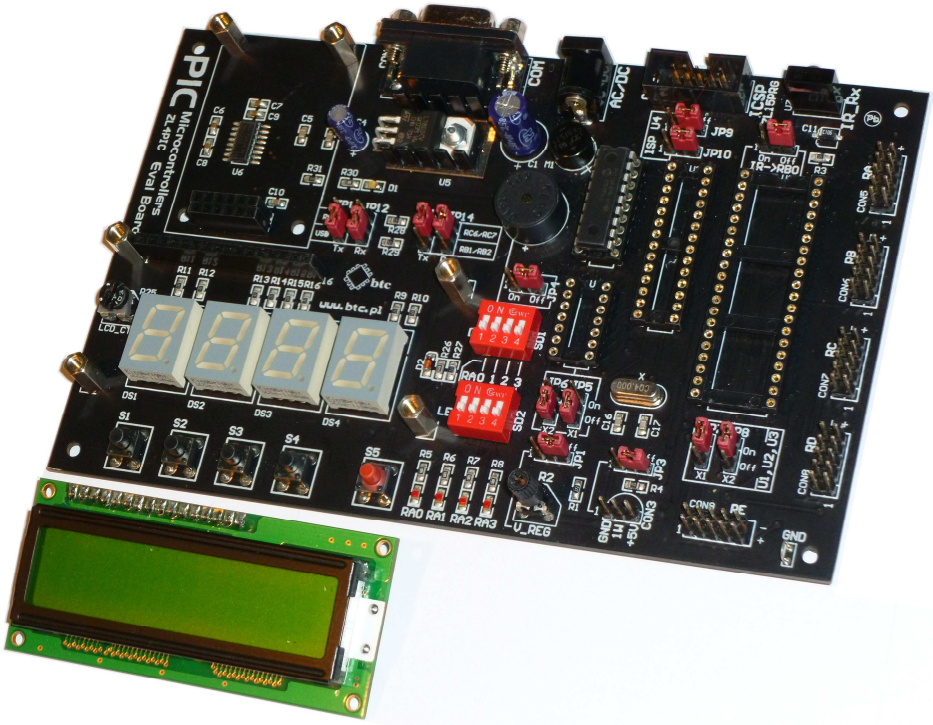
```
   list p=16f819, r=hex
#include p16f819.inc

; Jeśli zdefiniowana stała Internal_osc, to konfiguracja z wewn. oscylatorem RC,
; w przeciwnym razie użyj zewnętrznego rezonatora kwarcowego w trybie XT
#define Internal_osc

#ifdef Internal_osc
__config _CP_OFF & _CPD_OFF & _LVP_OFF & _BODEN_ON & _MCLR_ON & _PWRTE_ON &
_WDT_OFF & _INTRC_IO
#else
__config _CP_OFF & _CPD_OFF & _LVP_OFF & _BODEN_ON & _MCLR_ON & _PWRTE_ON &
_WDT_OFF & _XT_OSC
#endif

; Definicje adresów zmiennych
wyj_Q_nieQ          equ 0x20

; Inicjalizacja rejestrów
#ifdef Internal_osc
   banksel OSCCON            ; bank 1
   movlw   0x40
   iorwf   OSCCON,f          ; ustaw taktowanie wewnętrznego zegara RC na 1 MHz
#endif

   banksel ADCON1            ; bank 1
   movlw   0x06
   movwf   ADCON1            ; ustaw linie RA0...RA5 jako cyfrowe

   ; OPTION_REG w banku 1
   bcf  OPTION_REG, NOT_RBPU ; włącz rezystory do +5V w porcie B

   ; TRISA i TRISB w banku 1
   movlw   b'11111100'
   movwf   TRISA             ; RA0 i RA1 jako wyjścia, reszta portu B jako wejścia
   movlw   0xFF
   movwf   TRISB             ; wszystkie linie portu A ustaw jako wejścia

   banksel PORTA             ; bank 0
   movlw   b'01'
   movwf   wyj_Q_nieQ
   movwf   PORTA             ; ustaw początkowy stan wyjść RA0 (Q) i RA1 (nieQ)

main_loop
   ; W pętli wszystkie adresy w banku 0

   ; Przepisz bity nr 0 i 1 z wyj_Q_nieQ do rej. W z odwróceniem ich kolejności
   clrw
   btfsc   wyj_Q_nieQ,0      ; jeśli ustawiony bit 0,
   iorlw   b'10'             ;   to ustaw bit 1 w rej. W
   btfsc   wyj_Q_nieQ,1      ; jeśli ustawiony bit 1,
   iorlw   b'01'             ;   to ustaw bit 0 w rej. W

   ; Symulacja dwóch bramek NAND w przerzutniku RS
   ;    Q := NOT(nieR AND nieQ),
   ; nieQ := NOT(nieS AND Q),
   ; gdzie nieR i nieS sa bitami odpowiednio 0 i 1 odczytanymi z PORTB
   andwf   PORTB, W
   xorlw   b'11'             ; zaneguj tylko bity nr 0 i 1
   ; wynik w rej. W

   movwf wyj_Q_nieQ          ; Zachowaj wynik dla następnego obiegu pętli
   movwf PORTA               ; Ustaw wyjścia RA0 (Q) i RA1 (nieQ)

   goto    main_loop

   end
```

4. Dostępna aparatura

4.1. Zestaw uruchomieniowy

Zestaw ZL4PIC umożliwia uruchamianie programów na mikrokontrolerach z rodziny PIC12, PIC16 i niektórych PIC18 w obudowach DIP8, DIP14, DIP18, DIP28 i DIP40. Wygląd zestawu przedstawiono na rys. 3, natomiast schemat elektryczny na rys. F.1 w Aneksie F. Zestaw wyposażono m.in. w następujące peryferia:

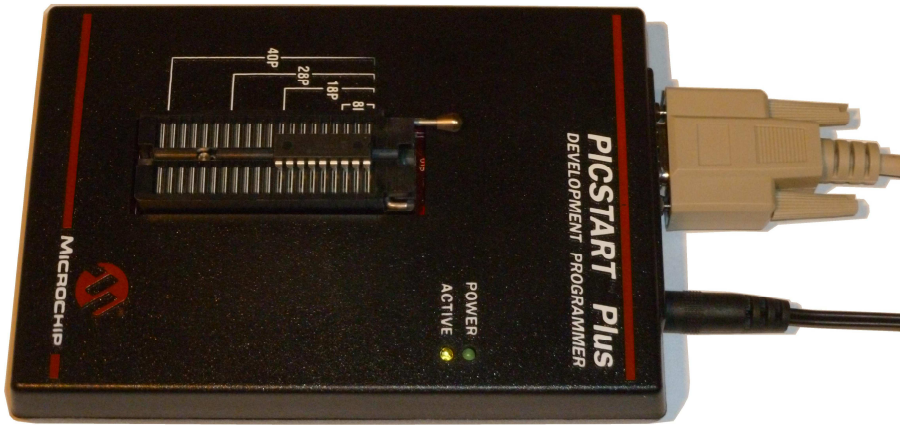
- cztery numeryczne wyświetlacze siedmiosegmentowe,
- wyświetlacz LCD 2×16 znaków, który można zamontować ponad wyświetlaczem numerycznym (oba typy wyświetlaczy nie mogą być jednocześnie używane),
- cztery przyciski ogólnego przeznaczenia S1,…,S4 podłączone do wejść mikrokontrolera RA0…RA3.
- przycisk resetu mikrokontrolera S5 dołączony do wejścia $\overline{\text{MCLR}}$, w przypadku niektórych mikrokontrolerów (np. PIC16F819) może być użyty także jako dodatkowy przycisk funkcyjny,
- cztery diody świecące, które w zależności od potrzeb mogą być przyłączone przełącznikiem SD2 do wejść mikrokontrolera RB0…RB3,
- przetwornik piezoceramiczny do sygnalizacji akustycznej,
- potencjometr V_REG do regulacji napięcia na wejściu przetwornika A/C dostępnego w niektórych mikrokontrolerach (np. PIC16F819, PIC16F877A),
- port RS232 umożliwiający komunikację z np. komputerem,
- odbiornik promieniowania podczerwonego, np. do odbioru sygnałów z pilota zdalnego sterowania,
- złącze magistrali 1Wire.



Rys. 3. Zestaw uruchomieniowy ZL4PIC ze zdjętym wyświetlaczem LCD.

4.2. Programator mikrokontrolerów

Przygotowany samodzielnie program przed uruchomieniem go w zestawie ZL4PIC należy zapisać do pamięci mikrokontrolera przy użyciu programatora PICSTART Plus. Programator ten jest przeznaczony do współpracy z pakietem MPLAB zainstalowanym na komputerze z systemem Windows. Komunikacja programatora z komputerem odbywa się przez złącze RS232. W przypadku braku takiego złącza w komputerze programator należy podłączyć za pośrednictwem konwertera RS232 ↔ USB.



Rys. 4. Programator PICSTART Plus z mikrokontrolerem zaciśniętym w podstawce.

4.3. Zasilacze

Na stanowisku doświadczalny wykorzystywane są dwa zasilacze:

1. Zasilacz 9V DC HiTRON model HES10-09007-0-7 dedykowany wyłącznie do pracy z programatorem PICSTART Plus.
2. Uniwersalny zasilacz dogniazdkowy z wielopozycyjnym przełącznikiem napięcia DC. Przed rozpoczęciem pracy z zastawem ZL4PIC zalecane jest wybranie w zasilaczu napięcia 9V. Napicie to jest obniżane do 5V przez stabilizator wbudowany w zestaw ZL4PIC.

UWAGA: wtyki wymienionych zasilaczy są zgodne, jednakże nie należy podłączać zasilacza HiTRON w sposób niezgodny z jego przeznaczeniem.

4.4. Komputer

W pracowni udostępniono komputery z systemem Windows i zainstalowanym pakietem MPLAB 8.66.

5. Przebieg ćwiczenia

W pierwszej kolejności należy przygotować do pracy zestaw złożony z komputera klasy PC i programatora PICSTART Plus połączonego z komputerem przez port RS232. Następnie należy przygotować w środowisku MPLAB IDE program dla jednego wybranego mikrokontrolera PIC z rodziny Midrange napisany w języku asembler. Program ten ma stanowić rozwiązanie problemu postawionego w treści wybranego zadania. Po pomyślnym skompilowaniu programu należy go zapisać w pamięci mikrokontrolera, a następnie przetestować w zestawie uruchomieniowym ZL4PIC. Opcjonalnie program można wypróbować najpierw w programowym symulatorze mikrokontrolera MPLAB SIM, a następnie przy wykorzystaniu rozwiązań sprzętowych.

5.1. Kolejność czynności

1. Podłączyć programator PICSTART Plus do komputera. Podczas tej czynności komputer powinien być wyłączony. Podłączyć zasilanie programatora napięciem stałym 9V – zalecane jest używanie w tym celu wyłącznie dedykowanego zasilacza HiTRON model HES10-09007-0-7.
2. Przygotować drugi zasilacz do pracy z zestawem uruchomienionym ZL4PIC. W tym celu wybrać przełącznikiem w zasilaczu napięcie 9V. Założyć odpowiednią końcówkę na przewód wyjściowy zasilacza, która pasuje do gniazda zasilania w zestawie ZL4PIC, przy czym biegunowość napięcia wyjściowego nie ma znaczenia. Nie podłączać jeszcze napięcia zasilającego zestaw ZL4PIC.
3. Uruchomić komputer i po załadowaniu systemu uruchomić środowisko MPLAB IDE.
4. Przed nawiązaniem łączności z programatorem należy wskazać jego typ w pakiecie MPLAB IDE. W tym celu wybieramy z menu pozycję `Programmer/Select Programmer` i dalej na liście model dostępny w pracowni `PICSTART Plus`. Następnie aktywujemy programator wybierając z menu `Programmer/Enable Programmer`. Jeżeli ta operacja nie powiodła się, to należy sprawdzić: zasilanie programatora (w programatorze powinna świecić się zielona dioda „POWER”), połączenie z komputerem przez port RS232 oraz numer portu szeregowego ustawiony w pakiecie MPLAB IDE w oknie dialogowym `Programmer` otwieranym przez wybranie z menu pozycji `Programmer/Settings`.
5. W porozumieniu z prowadzącym zajęcia wybrać zadanie, które będzie realizował mikrokontroler w zestawie uruchomieniowym ZL4PIC. Przykładowe zadania zebrano w rozdziale 5.2. Wykonawcy ćwiczeń mogą zaproponować własne zadania.
6. Wybrać jeden typ mikrokontrolera do zaprogramowania i zgłosić decyzję obsłudze pracowni. W pracowni powinny być dostępne przynajmniej mikrokontrolery typu PIC16F84A-04, PIC16F819 oraz PIC16F877A. Rozważyć czy wybrany typ mikrokontrolera będzie odpowiedni do realizacji wybranego zadania.
7. W celu rozpoczęcia edycji tekstu źródłowego nowego programu wybrać z menu pozycję `File/New`. Użycie asemblera MPASM dostępnego w środowisku MPLAB IDE nie wymaga utworzenia projektu (utworzenie projektu jest konieczne w przypadku wykorzystywania kompilatorów innych języków). Dalej założymy, że projekt nie będzie tworzony, a cały tekst programu będzie zawarty w jednym pliku.

UWAGA: deklarację typu mikrokontrolera w programie komendą „`list p=`” należy traktować raczej jako informację dla programisty niż wiążące polecenie dla asemblera (dotyczy np. MPLAB IDE wersja 8.66). Aby zapewnić wygenerowanie poprawnego kodu programu i prawidłową współpracę z programatorem należy wybrać typ mikrokontrolera także na liście w oknie dialogowym `Select Device` otwieranym przez wybranie z menu pozycji `Configure/Select Device...`.

Jeżeli bity konfiguracyjne nie są ustawiane w tekście źródłowym programu komendą „`__config`”, to trzeba je ustawić przed kompilacją w oknie dialogowym `Configuration Bits` otwieranym po wybraniu z menu `Configure/Configuration Bits...`. Nie należy bezkrytycznie akceptować domyślnych ustawień przyjętych w pakiecie MPLAB IDE, gdyż mogą one być nieodpowiednie.

8. Przygotować tekst źródłowy programu. Przed główną pętlą programu należy pamiętać o ustawieniu odpowiedniego stanu początkowego rejestrów OSCCON oraz ADCON1 (o ile istnieją w wybranym mikrokontrolerze), a także o określeniu kierunków transmisji danych przez porty wej./wyj. (rejestry TRISA i TRISB) oraz o ustawieniu początkowego stanu wyjść (rejestry PORTA i PORTB). Jeżeli będą wykorzystywane przyciski dostępne w zestawie ZL4PIC, to należy także włączyć na wejściach portu B rezystory podciągające napięcie do +5V (rejestr OPTION_REG). Wymienione rejestry leżą w różnych bankach pamięci (mapy pamięci znajdują się w Aneksie D) i trzeba pamiętać o wybieraniu odpowiedniego banku, np. komendą „banksel” (opisaną w Aneksie C). Zestaw instrukcji mikrokontrolerów zamieszczono w Aneksie A, natomiast sposób wykonywania operacji na liniach portów A i B omówiono w rozdziałach 3.6 i 3.7 niniejszej instrukcji. Rozkłady wyprowadzeń mikrokontrolerów pokazano w Aneksie E. Układ połączeń linii portów A i B z dwustanowymi przyciskami oraz diodami DEL w zestawie uruchomieniowym ZL4PIC przedstawiono na schemacie w Aneksie F oraz w odrębnej dokumentacji przygotowanej przez producenta zestawu [8].
9. W celu skompilowania programu wybrać z menu `Project/QuickBuilt`. Poprawić wszystkie błędy zgłaszane przez kompilator (nie dotyczy uwag opisywanych przez kompilator jako „Warning”).
10. Poprawnie skompilowany program można uruchomić najpierw w programowym symulatorze. W tym celu wybrać z menu pozycję `Debugger/Select Tool/MPLAB SIM`. Jeżeli wcześniej został wybrany programator PICSTART Plus, to zostanie on teraz anulowany. Opis symulatora wykracza jednak poza ramy niniejszej instrukcji a jego użycie nie jest obowiązkowe.
11. Zapisać kod programu w pamięci mikrokontrolera. Jeżeli wcześniej był używany symulator, to programator należy ponownie wybrać z menu `Programmer/Select Programmer/PICSTART Plus`. Następnie aktywujemy programator wybierając z menu `Programmer/Enable Programmer`. Mikrokontroler należy włożyć do podstawki w programatorze i zacisnąć go dźwigienką. Nie ma potrzeby deaktywacji programatora przed włożeniem lub wyjęciem mikrokontrolera do/z podstawki programatora, gdyż napięcia na podstawce są załączane tylko na czas transmisji danych.
    UWAGA: mikrokontroler należy odpowiednio zorientować względem podstawki, tzn. nóżka mikrokontrolera o numerze `1` oznaczona kropką wytłoczoną w rogu jego obudowy musi trafić do złącza w podstawce opisanego cyfrą `1`.

Następnie wybrać z menu `Programmer/Program`. Jeżeli nie zostały zgłoszone żadne błędy, to mikrokontroler z zapisanym programem jest gotowy do użycia. Wyjąć mikrokontroler z programatora.

12. Włożyć mikrokontroler do podstawki z zestawie uruchomieniowym ZL4PIC zwracając uwagę na odpowiednią orientację układu scalonego względem podstawki: półokrągłe nacięcie na brzegu mikrokontrolera powinno znaleźć się po stronie analogicznego nacięcia w podstawce.
    UWAGA: pamiętać o odłączeniu zasilacza zestawu ZL4PIC przed każdym założeniem lub wyjęciem mikrokontrolera.
13. Przed załączeniem zasilania odpowiednio skonfigurować zestaw ZL4PIC wykorzystując schemat zamieszczony w Aneksie F. W szczególności należy:
    - Odłączyć potencjometr od linii mikrokontrolera RA0 przez ustawienie zworki JP1 w pozycji Off.
    - Odłączyć odbiornik podczerwieni od linii mikrokontrolera RB0 przez ustawienie zworki JP2 w pozycji Off.
    - Odłączyć brzęczyk piezoceramiczny, przez ustawienie zworki JP4 w pozycji Off.
    - Rezonator kwarcowy dla mikrokontrolerów w obudowach zawierających 18, 28 oraz 40 wyprowadzeń przyłącza się przez ustawienie zworek JP7 i JP8 w pozycji On (podłączony pin 1). W przypadku, gdy używany jest wewnętrzny oscylator RC, zworki te muszą być ustawione w pozycji Off.
    - Wyjąć wyświetlacz LCD.
    - Odłączyć linie mikrokontrolera RB1 i RB2 od konwertera napięć w układzie scalonym MAX232. W tym celu zworki JP13 i JP14 ustawić w pozycji RC6/RC7.
    - Odłączyć tranzystory sterujące wyświetlaczem numerycznym LED przez ustawienie wszystkich mikroprzełączników w sekcji SD1 w pozycji Off.
    - Przyłączyć diody świecące LED przez ustawienie mikroprzełączników w sekcji SD2 w pozycji On.
14. Podłączyć zasilanie zestawu ZL4PIC. Zalecane jest podłączenie najpierw samego zasilacza do gniazda sieci 230V~ a następnie włożenie wtyczki pod napięciem 9V do gniazda oznaczonego AC/DC w zestawie ZL4PIC. W przypadku wykonania czynności w odwrotnej kolejności niektóre egzemplarze mikrokontrolerów mogą nie uruchomić się z powodu zbyt długiego czasu narastania napięcia zasilającego.
15. Przetestować działanie mikrokontrolera w zestawie ZL4PIC. Jeżeli program nie działa zgodnie z założeniami, wyciągnąć wnioski i wykonać poprawki w programie. Rozważyć konsultacje z obsługą pracowni.
16. Po osiągnięciu zgodności działania programu z złożeniami wykonać krótką demonstrację przed obsługą laboratorium.
17. Zakończenie pracy.
    a. Zdeaktywować programator wybierając z menu w pakiecie MPLAB IDE `Programmer/Disable Programmer`.
    b. Zamknąć pogram MPLAB.
    c. Odłączyć przewody zasilania programatora i zestawu ZL4PIC oraz wyłączyć oba zasilacze z kontaktów sieci 230V~.
    d. Wykonać kopię plików związanych z projektem na własnym nośniku danych.
    e. Jeżeli obsługa pracowni zaleci uprzątnięcie stanowisk to należy wyłączyć komputer, a dopiero w drugiej kolejności odłączyć przewód z portów RS232 w komputerze i programatorze. Następnie zapakować urządzenia i przewody do odpowiednich pudełek.

5.2. Propozycje zadań realizowanych przez mikrokontroler

Należy zaprogramować mikrokontroler do realizacji wybranego zadania oraz przetestować jego działanie w zestawie uruchomieniowym ZL4PIC. Jako wejścia wykorzystać przyciski S1 ÷ S4 (ewentualnie także wejście resetu S5); jako wyjścia wykorzystać diody świecące LED oznaczone RA0 ÷ RA3. Liczba gwiazdek w nawiasach opisuje stopień trudności zadania. **Uwaga:** w zestawie ZL4PIC na wejściach obowiązuje logika ujemna, tzn. przyciśnięcie przycisku opowiada stanowi 0 na wejściu mikrokontrolera, natomiast na wyjściach obowiązuje logika dodatnia, tzn. diody świecą się przy stanie 1 na wyjściu mikrokontrolera.

1. (*) Symulacja układu kombinacyjnego zamieniającego 4-bitowe liczby w kodzie NKB na 4-bitowe liczby w kodzie Graya BRGC. Przytrzymywanie przycisków S4…S1 w zestawie ZL4PIC należy zinterpretować jako wartość 1 odpowiedniego bitu liczby NKB. Liczbę w kodzie Graya wyświetlić przy użyciu diod LED.
2. (*) Układ, który po wykryciu przyciśnięcia *i*-tego przycisku zapala *i*-tą diodę świecącą LED i gasi inne diody. Podczas zwalniania przycisku stan diod pozostaje bez zmian. Liczba aktywnych przycisków powinna wynosić 3 albo 4. Początkowo wszystkie diody powinny być zgaszone.
3. (**) Układ zapalający kolejno po jednej diodzie DEL spośród diod opisanych w zestawie ZL4PIC symbolami RA0, RA1 i RA2. Po naciśnięciu przycisku S1 światło powinno przesuwać się cyklicznie w prawo, tzn. diody zapalają się w kolejności RA0, RA1 i RA2, natomiast po przyciśnięciu S2 w lewo, tzn. w kolejności RA2, RA1, RA0. Zalecany czas świecenia jednej diody wynosi około 0,5 s.
4. (**) Układ eliminacji drgań styków w jednym przycisku S1 zestawu ZL4PIC. Po stwierdzeniu przyciśnięcia przycisku, tzn. przejścia $1 \rightarrow 0$ na wejściu mikrokontrolera, stan diody LED o symbolu RA0 należy zmienić na przeciwny w taki sposób, by nie dochodziło do wielokrotnych przełączeń na skutek drgań styków. *Wskazówka*: jednym z możliwych rozwiązań jest natychmiastowe przełączenie stanu diody po wykryciu przejścia $1 \rightarrow 0$, a następnie ignorowanie przez pewien dobrany eksperymentalnie czas wszelkich przełączeń stanu przycisku. Ponieważ użycie sprzętowych zegarów wbudowanych w mikrokontroler wykracza poza ramy tego ćwiczenia, opóźnienie można zaimplementować przy wykorzystaniu pętli o zadanej liczbie powtórzeń bez wykonywania dodatkowych czynności. Rozważyć czasy opóźnień rzędu 1…10 ms.
5. (**) Układ generujący 4-bitowe liczby o wysokim stopniu losowości na podstawie czasu przytrzymywania przycisku S1 w zestawie ZL4PIC. Liczby są wyświetlane binarnie przy użyciu czterech diod LED.
6. (**) Układ 3-bitowego rejestru zatrzaskowego wyzwalanego poziomem 0. Podczas przytrzymywania przycisku S4, który w zestawie ZL4PIC jest połączony z wejściem mikrokontrolera RB3 stan logiczny wejść jest przepisywany z negacją na wyjścia diod świecących RA0, RA1 i RA2. Po zwolnieniu przycisku S4 stan wyjść zostaje zachowany bez zmian (zatrzaśnięty). Negacja stanu wejść wynika z logiki ujemnej obowiązującej na wejściach połączonych z przyciskami.
7. (***) Układ, który po wykryciu przejścia $1 \rightarrow 0$ na *i*-tym wejściu włącza *i*-tą diodę bez gaszenia diod zapalonych wcześniej. Gdy świecą się już wszystkie diody i zostanie wykryte kolejne przejście $1 \rightarrow 0$ na dowolnym wejściu, to wszystkie diody są gaszone.
8. (***) Układ eliminacji drgań styków czterech przycisków S1 ÷ S4 w zestawie ZL4PIC. Po stwierdzeniu przyciśnięcia przycisku, tzn. przejścia $1 \rightarrow 0$ na jednym z wejść RB0, RB1, RB2 lub RB3 mikrokontrolera, stan diody LED o odpowiednim numerze (RA0, RA1, RA2 lub RA3) należy zmienić na przeciwny w taki sposób, by nie doszło do

wielokrotnych przełączeń na skutek drgań styków. Uwaga: w tym zadaniu wymagane jest aby opóźnienie odliczane dla jednego z wejść w celu eliminacji drgań styków nie blokowało wykrywania przyciśnięć przycisków na pozostałych wejściach.

9. (****) Zasymulować jeden przerzutnik JK-MS z dodatkowymi wejściami asynchronicznego zerowania i ustawiania $\overline{PR}$ i $\overline{CLR}$. Ponieważ taki przerzutnik ma w sumie 5 wejść, należy wykorzystać także przycisk resetu S5 do symulacji wejścia $\overline{CLR}$. Uwaga: w mikrokontrolerach PIC16F84A oraz PIC16F877A wejście mikrokontrolera połączone z przyciskiem S5 może pracować tylko jako wejście resetu $\overline{MCLR}$. W mikrokontrolerze PIC16F819 może pracować zarówno jako $\overline{MCLR}$ jak i uniwersalny port wej./wyj. w zależności od stanu rejestru konfiguracyjnego.

6. Wskazówki do raportu

Raport powinien zawierać:

1. Stronę tytułową (wg wzoru).
2. Sformułowanie celu ćwiczenia.
3. Wykaz użytej aparatury. W szczególności należy pamiętać o podaniu symbolu katalogowego wybranego mikrokontrolera.
4. Treść zadania, które powinien realizować zaprogramowany mikrokontroler w zestawie ZL4PIC.
5. Jeżeli treść zadania nie jest jednoznaczna, podać przyjęte dodatkowe założenia projektowe.
6. Opis funkcji programu poprzez algorytm działania. Alternatywnie, można rozważyć układ kombinacyjny albo sekwencyjny równoważny zaprogramowanemu mikrokontrolerowi i przedstawić odpowiednie tablice prawdy albo tablice przejść i wyjść dla każdego wyjścia.
7. Konfigurację wybranych zworek i przełączników w zestawie ZL4PIC o krytycznym znaczeniu dla działania programu.
8. Tekst źródłowy programu.
9. Dyskusję uzyskanych wyników. W szczególności rozważyć:
   a). Czy uzyskane wyniki są zgodne z założeniami. W przypadku wystąpienia rozbieżności opisać środki podjęte w celu ich usunięcia, znalezione błędy i uzyskany ostatecznie rezultat.
   b). Czy wystąpiły jakieś problemy w wykorzystywanym sprzęcie, oprogramowaniu lub w dokumentacji?
   c). Czy z perspektywy czasu można dostrzec jakieś możliwości uproszczenia lub poprawy algorytmu działania?
   d). Czy wybrany algorytm można by zaimplementować w jakiś alternatywny sposób, np. (a) implementacja oparta na operacjach logicznych NOT, AND, OR, XOR w sposób naśladujący połączenia bramek logicznych; (b) tablice prawdy/przejść/wyjść zrealizowane dosłownie jako tablice w pamięci RAM lub w kodzie programu; (c) wykorzystanie (pseudo)instrukcji skoków warunkowych do podejmowania decyzji?

W raporcie ocenie podlegać będzie obecność, poprawność i jakość wszystkich wymienionych powyżej składników. Wstęp teoretyczny nie jest wymagany i w przypadku jego zamieszczenia w raporcie nie wpłynie na ocenę.

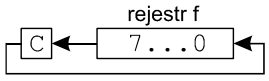
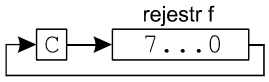
7. Literatura

[1] S. Pietraszek, *Mikroprocesory jednoukładowe PIC*, Helion, Gliwice 2002.
[2] T. Jabłoński, *Mikrokontrolery PIC16F8x w praktyce*, Wydawnictwo BTC, Warszawa 2002.
[3] *MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User's Guide*, Data Sheet DS33014K, Microchip Technology Inc. 2009, dostępne na stronie www.microchip.com.
[4] *PICmicro™ Mid-Range MCU Family Reference Manual*, Data Sheet DS33023A, Microchip Technology Inc. 1997, dostępne na stronie www.microchip.com.
[5] *PIC16F84A*, Data Sheet DS35007C, Microchip Technology Inc. 2001-2013, dostępne na stronie www.microchip.com.
[6] *PIC16F818/819*, Data Sheet DS39598F, Microchip Technology Inc. 2001-2013, dostępne na stronie www.microchip.com.
[7] *PIC16F87XA*, Data Sheet DS39582C, Microchip Technology Inc. 2001-2013, dostępne na stronie www.microchip.com.
[8] *ZL4PIC Uniwersalny zestaw uruchomieniowy dla mikrokontrolerów PIC*, Wydawnictwo BTC 2005, dostępne na stronie www.fizyka.p.lodz.pl/pl/dla-studentow/tc/.

Aneksy

A. Lista rozkazów mikrokontrolerów PIC z rodziny Midrange

Tabela A.1. Rozkazy operujące na bajtach.

Mnemonik, argument	Opis	Funkcja	Zmieniane znaczniki	Kod rozkazu
ADDWF f,d	Dodaj W i f	W + f → d	C, DC, Z	`00 0111 dfff ffff`
ANDWF f,d	Bitowy iloczyn logiczny W i f	W AND f → d	Z	`00 0101 dfff ffff`
CLRF f	Wyzeruj f	0 → f	Z	`00 0001 1fff ffff`
CLRW	Wyzeruj W	0 → W	Z	`00 0001 0xxx xxxx`
COMF f,d	Zaneguj f	NOT f → d	Z	`00 1001 dfff ffff`
DECF f,d	Zmniejsz f o 1	f – 1 → d	Z	`00 0011 dfff ffff`
DECFSZ f,d	Zmniejsz f o 1, pomiń następny rozkaz jeśli wynik=0	f – 1 → d, jeśli d=0 to nic nie rób podczas następnego rozkazu	–	`00 1011 dfff ffff`
INCF f,d	Zwiększ f o 1	f + 1 → d	Z	`00 1010 dfff ffff`
INCFSZ f,d	Zwiększ f o 1, pomiń następny rozkaz jeśli wynik=0	f + 1 → d, jeśli d=0 to nic nie rób podczas następnego rozkazu	–	`00 1111 dfff ffff`
IORWF f,d	Bitowa suma logiczna W i f	W OR f → d	Z	`00 0100 dfff ffff`
MOVF f,d	Prześlij f do d	f → d	Z	`00 1000 dfff ffff`
MOVWF f	Prześlij W do f	W → f	–	`00 0000 1fff ffff`
NOP	Nic nie rób	–	–	`00 0000 0xx0 0000`
RLF f,d	Przesuń f cyklicznie w lewo przez znacznik C		C	`00 1101 dfff ffff`
RRF f,d	Przesuń f cyklicznie w prawo przez znacznik C		C	`00 1100 dfff ffff`
SUBWF f,d	Odejmij W od f	f – W → d	C,DC,Z	`00 0010 dfff ffff`
SWAPF f,d	Zamień tetrady w f	f<0:3> ↔ f<4:7>	–	`00 1110 dfff ffff`
XORWF f,d	Bitowa suma modulo 2 na W i f	W XOR f → d	Z	`00 0110 dfff ffff`

Oznaczenia:
d – bit wyboru przeznaczenia wyniku:
d=0: zapisz wynik w rejestrze W,
d=1: zapisz wynik w pamięci pod adresem f,
f – 7-bitowy adres w pamięci,
bity znaczników omówiono w Aneksie B. Rejestr STATUS.

Tabela A.2. Rozkazy operujące na bitach.

Mnemonik, argument	Opis	Funkcja	Zmieniane znaczniki	Kod rozkazu
BCF f,b	Wyzeruj bit b w bajcie f	0 → f<b>	–	01 00bb bfff ffff
BSF f,b	Ustaw bit b w bajcie f	1 → f<b>	–	01 01bb bfff ffff
BTFSC f,b	Testuj bit b w bajcie f, jeżeli wyzerowany to pomiń następny rozkaz	jeśli f<b>=0, to nic nie rób podczas następnego rozkazu	–	01 10bb bfff ffff
BTFSS f,b	Testuj bit b w bajcie f, jeżeli ustawiony to pomiń następny rozkaz	jeśli f<b>=1, to nic nie rób podczas następnego rozkazu	–	01 11bb bfff ffff

Oznaczenia:
b – 3-bitowy numer bitu w bajcie pod adresem f,
f – 7-bitowy adres w pamięci,
f<b> – bit nr b w bajcie o adresie f.

Tabela A.3. Rozkazy operujące na stałych.

Mnemonik, argument	Opis	Funkcja	Zmieniane znaczniki	Kod rozkazu
ADDLW k	Dodaj W i stałą k	W + k → W	C, DC, Z	11 111x kkkk kkkk
ANDLW k	Bitowy iloczyn logiczny W i stałej k	W AND k → W	Z	11 1001 kkkk kkkk
IORLW k	Bitowa suma logiczna W i stałej k	W OR k → W	Z	11 1000 kkkk kkkk
MOVLW k	Prześlij stałą k do W	k → W	–	11 00xx kkkk kkkk
RETLW k	Powrót z podprogramu ze stałą k w rej. W	k → W, TOS → PC	–	11 01xx kkkk kkkk
SUBLW k	Odejmij W od stałej k	k – W → W	C, DC, Z	11 110x kkkk kkkk
XORLW k	Bitowa suma modulo 2 na W i stałej k	W XOR k → W	Z	11 1010 kkkk kkkk

RETLW – ten rozkaz uwzględniono także w grupie rozkazów sterujących.

Oznaczenia:
k – 8-bitowa stała,
PC – licznik rozkazów (*program counter*),
TOS – 13-bitowy adres na wierzchołku stosu (*top of stack*),
bity znaczników omówiono w Aneksie B. Rejestr STATUS.

Tabela A.4. Rozkazy sterujące.

Mnemonik, argument	Opis	Funkcja	Zmieniane znaczniki	Kod rozkazu
CALL a	Wywołaj podprogram	PC + 1 → TOS, a → PC<10:0> PCLATH<4:3> → PC<12:11>	–	10 0aaa aaaa aaaa
CLRWDT	Wyzeruj licznik „Watchdoga”	0 → WDT	$\overline{TO}$, $\overline{PD}$	00 0000 0110 0100
GOTO a	Skok bezwarunkowy	a → PC<10:0> PCLATH<4:3> → PC<12:11>	–	10 1aaa aaaa aaaa
OPTION	Zapisz rej. OPTION_REG	W → OPTION_REG	–	00 0000 0110 0010
RETFIE	Powrót z procedury obsługi przerwania	TOS → PC, 1 → GIE	–	00 0000 0000 1001
RETLW k	Powrót z podprogramu ze stałą k w rej. W	k → W, TOS → PC	–	11 01xx kkkk kkkk
RETURN	Powrót z podprogramu	TOS → PC	–	00 0000 0000 1000
SLEEP	Przejdź w tryb uśpienia	0 → WDT, zatrzymanie oscylatora	$\overline{TO}$, $\overline{PD}$	00 0000 0110 0011
TRIS r	Zapisz jeden z rejestrów TRISr	W → TRISr (rejestr nr r w banku 1)	–	00 0000 0110 0rrr

Oznaczenia:
a – 11-bitowa stała adresowa,
k – 8-bitowa stała,
r – 3-bitowy adres rejestru w 1 banku pamięci,
r = 5: rejestr TRISA; r = 6: rejestr TRISB; r = 7: rejestr TRISC (o ile istnieje port wej./wyj. C),
GIE – bit zezwolenia na niezamaskowane przerwania (przerwania włączone gdy =1) w rejestrze INTCON,
PC – licznik rozkazów (*program counter*),
TOS – 13-bitowy adres na wierzchołku stosu (*top of stack*).

Uwaga:
Rozkazy OPTION i TRIS zostały uznane przez producenta za przestarzałe i mogą zostać usunięte z listy rozkazów w nowo opracowanych mikrokontrolerach. Rozkazy OPTION i TRIS gwarantują zapis do odpowiedniego rejestru niezależnie od aktualnie wybranego banku pamięci. Alternatywnie rejestry OPTION_REG i TRISx można zapisać instrukcją MOVWF i odczytywać instrukcją MOVF przy odpowiednio ustawionym banku pamięci, np.:

```
banksel  TRISB
MOVWF    TRISB
```

W asemblerze MPASM zaimplementowano szereg użytecznych predefiniowanych makropoleceń, które można traktować jak dodatkowe rozkazy mikrokontrolera.

Tabela A.5. Pseudoinstrukcje.

Mnemonik, argument	Opis	Równoważne operacje	Zmieniane znaczniki
ADDCF f,d	Dodaj przeniesienie C do f, zapisz wynik do d	BTFSC 3, 0 INC f, d	Z
ADDDCF f,d	Dodaj przeniesienie połówkowe DC do f, zapisz wynik do d	BTFSC 3, 1 INCF f, d	Z
BC a	Skok do etykiety a gdy C = 1	BTFSC 3, 0 GOTO a	–
BDC a	Skok do etykiety a gdy DC = 1	BTFSC 3, 1 GOTO a	–
BNC a	Skok do etykiety a gdy C = 0	BTFSS 3, 0 GOTO a	–
BNDC a	Skok do etykiety a gdy DC = 0	BTFSS 3, 1 GOTO a	–
BNZ a	Skok do etykiety a gdy Z = 0	BTFSS 3, 2 GOTO a	–
BZ a	Skok do etykiety a gdy Z = 1	BTFSC 3, 2 GOTO a	–
CLRC	Wyzeruj znacznik C	BCF 3, 0	C
CLRDC	Wyzeruj znacznik DC	BCF 3, 1	DC
CLRZ	Wyzeruj znacznik Z	BCF 3, 2	Z
LCALL a	Dalekie wywołanie procedury (między segmentami)	BCF / BSF 0x0A, 3 BCF / BSF 0x0A, 4 CALL a	–
LGOTO a	Daleki skok (między segmentami)	BCF / BSF 0x0A, 3 BCF / BSF 0x0A, 4 GOTO a	–
MOVFW f	przepisz f → W	MOVF f, 0	Z
NEGF f,d	Negacja arytmetyczna f (uzupełnienie do 2), zapisz wynik do d	COMF f, 1 INCF f, d	Z
SETC	Ustaw znacznik C	BSF 3, 0	C
SETDC	Ustaw znacznik DC	BSF 3, 1	DC
SETZ	Ustaw znacznik Z	BSF 3, 2	Z
SKPC	Omiń następną instrukcję gdy C = 1	BTFSS 3, 0	–
SKPDC	Omiń następną instrukcję gdy DC = 1	BTFSS 3, 1	–
SKPNC	Omiń następną instrukcję gdy C = 0	BTFSC 3, 0	–
SKPNDC	Omiń następną instrukcję gdy DC = 0	BTFSC 3, 1	–
SKPNZ	Omiń następną instrukcję gdy Z = 0	BTFSC 3, 2	–
SKPZ	Omiń następną instrukcję gdy Z = 1	BTFSS 3, 2	–
SUBCF f,d	Odejmij przeniesienie C od f, zapisz wynik do d	BTFSC 3, 0 DECF f, d	Z
SUBDCF f,d	Odejmij przeniesienie połówkowe DC od f, zapisz wynik do d	BTFSC 3, 1 DECF f, d	Z
TSTF f	Testuj rejestr f, jeżeli zero to 1 → Z, jeżeli różny od zera to 0 → Z	MOVF f, 1	Z

Bity znaczników omówiono w Aneksie B. Rejestr STATUS.

B. Rejestr STATUS

Rejestr STATUS zawiera znaczniki wyniku operacji w jednostce arytmetyczno-logicznej (ALU), bity wyboru banku pamięci danych oraz bity przyczyny (re)startu mikrokontrolera.

Rejestr STATUS, adres 03h

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
bit 7							bit 0

Dotyczy wszystkich układów PIC16 z rodziny Midrange

Oznaczenia:			
R = bit do odczytu,	W = bit do zapisu,		
„- n” = wartość po włączeniu zasilania (POR):	„1” = bit ustawiony,	„0” = bit wyzerowany,	„x” = stan dowolny.

Opisy bitów:

bit 7 **IRP**: numer banku pamięci danych dla adresowania pośredniego
0 = banki 0 i 1 (adresy 00h…FFh),,
1 = banki 2 i 3 (adresy 100h – 1FFh).

bit 6-5 **RP<1:0>**: numer banku pamięci danych dla adresowania bezpośredniego
00 = bank 0 (adresy 00h…7Fh),
01 = bank 1 (adresy 80h…FFh),
10 = bank 2 (adresy 100h…17h),
11 = bank 3 (adresy 180h…1FFh).

bit 4 $\overline{\textbf{TO}}$ (Time-out): znacznik przepełnienia licznika WDT (Watchdog Timer)
1 = po włączeniu zasilania, wykonaniu instrukcji CLRWDT lub SLEEP,
0 = po przepełnieniu się licznika WDT.

bit 3 $\overline{\textbf{PD}}$: (Power-down): znacznik uśpienia mikrokontrolera
1 = po włączeniu zasilania lub wykonaniu instrukcji CLRWDT,
0 = po wykonaniu instrukcji SLEEP.

bit 2 **Z** (Zero): bit zerowego wyniku operacji arytmetyczno-logicznych
1 = rezultat operacji jest zerowy,
0 = rezultat operacji jest różny od zera.

bit 1 **DC** (Digit Carry): bit przeniesienia/pożyczki z dolnej tetrady (połówki bajtu)
używany w operacjach arytmetycznych na kodzie BCD
(instrukcje ADDLW, ADDWF, SUBLW, SUBWF)
1 = nastąpiło przeniesienie z najbardziej znaczącego bitu dolnej tetrady,
0 = nie ma przeniesienia z najbardziej znaczącego bitu dolnej tetrady.

bit 0 **C** (Carry): bit przeniesiena/pożyczki w operacjach arytmetycznych
(instrukcje ADDLW, ADDWF, SUBLW, SUBWF)
1 = nastąpiło przeniesienie z najbardziej znaczącego bitu wyniku,
0 = nie ma przeniesienia z najbardziej znaczącego bitu wyniku.
Ponadto C jest 9-tym bitem w przesunięciach cyklicznych (instrukcje RRF, RLF).

Uwaga 1: mikrokontroler PIC16F84A posiada tylko dwa banki pamięci danych i bity 7 oraz 6 w rejestrze STATUS nie są zaimplementowane.

Uwaga 2: instrukcje odejmowania SUBLW i SUBWF stosują odwróconą konwencję wartości znaczników C i DC: 1 – nie było pożyczki, 0 – wystąpiła pożyczka arytmetyczna.

C. Wybrane komendy asemblera MPASM

list – polecenie wielozadaniowe

Składnia: list [*opcja*, *opcja*, …]

Opis: Dokumentacja do asemblera MPASM wylicza 14 opcji o różnorodnych znaczeniach. W niniejszej instrukcji ograniczymy się do dwóch wybranych:

- p=type - określa typ mikrokontrolera, np. p=PIC16F84,
- r=podstawa - określa domyślny system liczbowy dla stałych bez jawnie podanego formatu; możliwe podstawy systemu liczbowego: hex, dec, oct.

Przykład: list p=16f819, r=hex - procesor typu PIC16F819, liczby w systemie szesnastkowym.

Komendy deklaracji danych w pamięci programu

__config – Ustawienie bitów konfiguracyjnych mikrokontrolera

Składnia: __config *expr*

Opis: Określa wartość 14-bitowego słowa konfiguracyjnego pod adresem 2007h w przestrzeni adresowej programu, która zostanie zapisana do mikrokontrolera razem z kodem programu. Patrz rozdział: „Pamięć konfiguracji w mikrokontrolerach PIC z rodziny Midrange” na początku instrukcji.

db – deklaracja jednego bajtu danych

Składnia: [etykieta] db *wyrażenie1* [, *wyrażenie2*, *wyrażenie3*, ...]

Opis: Rezerwuje 14-bitowe słowa w pamięci programu i wypełnia jedno słowo przez dwie kolejne wartości 8-bitowych wyrażeń, przy czym pierwszy argument na liście i kolejne nieparzyste argumenty są obcinane do 6 bitów. Jeżeli liczba argumentów jest nieparzysta, to ostatnie słowo zostanie dopełnione wartością zero. Wielokrotne wyrażenia wypełniają następujące po sobie słowa aż do zakończenia listy wyrażeń.

Przykład: db 0x0f, "e", 0x0f, "s", 0x0f, "t", "\n"
wypełnia kolejne słowa w pamięci programu wartościami: 0x0F65, 0x0F73, 0x0F74, 0x0a00.

dt – deklaracja tablicy instrukcji RETLW

Składnia: [etykieta] dt *wyrażenie1* [, *wyrażenie2*, *wyrażenie3*, ...]

Opis: Generuje serię instrukcji RETLW, po jednej instrukcji na każde wyrażenie. Wartość każdego wyrażenia musi mieścić się w 8-bitach. Każdy znak z łańcucha znaków jest zapisywany do swojej odrębnej instrukcji.

Przykład: dt "es", 0
generuje sekwencję instrukcji:

```
RETLW   0x65
RETLW   0x73
RETLW   0x00
```

dw – deklaracja jednego słowa danych

Składnia: [etykieta] dw *wyrażenie1* [, *wyrażenie2*, *wyrażenie3*, ...]

Opis: Rezerwuje 14-bitowe słowa w pamięci programu i wypełnia pojedyncze słowo przez wartość jednego wyrażenia. Wielokrotne wyrażenia wypełniają następujące po sobie słowa aż do zakończenia listy wyrażeń.

Przykład: dw 0x0f65, 0x0f73, 0x0f74, 0
wypełnia kolejne słowa w pamięci programu wartościami: 0x0F65, 0x0F73, 0x0F74, 0x0000.

Komendy kontroli przebiegu kompilacji

#define – definicja podstawienia tekstowego

Składnia: #define *nazwa* [*łańcuch_znaków*]

Opis: Ta komenda powoduje od następnej linii zastępowanie podanej nazwy przez wymieniony dalej łańcuch znaków. Podstawienia pustymi łańcuchami znaków są przydatne w połączeniu z komendami kompilacji warunkowej ifdef oraz ifndef. Łańcuch znaków może zawierać dowolne znaki, w tym także odstępy, stałe, wyrażenia i rozkazy.

#include – wstawienie zawartości pliku

Składnia include *nazwa_pliku*, include "*nazwa_pliku*", include <*nazwa_pliku*>, #include *nazwa_pliku*, #include "*nazwa_pliku*" lub #include <*nazwa_pliku*>

Opis Wstawienie tekstu z podanego pliku w miejsce wywołania tej komendy. Równoważny efekt można osiągnąć przez wklejenie całego tekstu z podanego pliku zamiast komendy include.

#undefine – odwołanie podstawienia tekstowego

Składnia #undefine *nazwa*

Opis Odwołanie w dalszej części programu podstawienia tekstowego zdefiniowanego wcześniej komendą #define.

constant – definicja stałej asemblerowej

Składnia constant *nazwa=wyrażenie*

Opis Definiuje stałą symboliczną do użytku w wyrażeniach asemblera. Zdefiniowana stała nie może zostać usunięta ani przedefiniowana w dalszej części programu. Wyrażenie musi być w pełni określone w chwili definiowania nazwy symbolicznej.

end – zakończenie programu

Składnia end

Opis Kończy kompilację programu. Wszystkie następujące dalej instrukcje zostaną zignorowane.

equ – definicja stałej asemblerowej

Składnia *nazwa* equ *wyrażenie*

Opis Przypisuje wartość podanego wyrażenia do nazwy stałej symbolicznej. Zdefiniowana stała nie może zostać usunięta ani przedefiniowana w dalszej części programu. Komenda equ typowo jest używana do definiowania nazw dla adresów danych w pamięci RAM.

org – ustawienie adresu początkowego

Składnia *etykieta* org *wyrażenie*

Opis Określa adres początkowy w pamięci programu dla instrukcji umieszczonych po org.

set – definicja zmiennej asemblerowej

Składnia *nazwa* set *wyrażenie*

Opis Przypisuje wartość podanego wyrażenia do nazwy symbolicznej. Definicja może być zmieniona w dalszej części programu przez ponowne wywołanie komendy set.

Kompilacja warunkowa

else – rozpoczęcie alternatywnego bloku kompilacji warunkowej

Składnia else lub #else

Opis Używane w połączeniu z komendami if, ifdef oraz ifndef. Jeżeli nie jest spełniony warunek kompilacji fragmentu występującego bezpośrednio po tych instrukcjach, to alternatywnie zostanie skompilowany fragment pomiędzy else oraz endif.

endif – zakończenie bloku kompilacji warunkowej

Składnia endif lub #endif

Opis Kończy blok kompilowany warunkowo, który rozpoczyna się komendą if, ifdef albo ifndef.

if – rozpoczęcie bloku kompilacji warunkowej

Składnia if *wyrażenie* lub #if *wyrażenie*

Opis Jeżeli *wyrażenie* jest prawdziwe, to zostanie skompilowany fragment programu umieszczony bezpośrednio po instrukcji if. W przeciwnym przypadku, fragment ten zostanie pominięty aż do najbliższej instrukcji else albo endif. Wyrażenia o wartości różnej od zera są interpretowane jako prawdziwe; równe zero jako fałszywe.

ifdef – kompilacja jeżeli symbol został zdefiniowany

Składnia ifdef *nazwa* lub #ifdef *nazwa*

Opis Jeżeli *nazwa* została wcześniej zdefiniowana komendą #define, to zostanie skompilowany fragment programu umieszczony bezpośrednio po instrukcji ifdef. Jeżeli *nazwa* nie została zdefiniowana, to zostanie pominięty fragment aż do najbliższej instrukcji else albo endif.

ifndef – kompilacja jeżeli symbol nie został zdefiniowany

Składnia ifndef *nazwa* lub #ifndef *nazwa*

Opis Jeżeli *nazwa* nie została wcześniej zdefiniowana lub definicja została anulowana komendą #undefine, to zostanie skompilowany fragment programu umieszczony bezpośrednio po instrukcji ifndef. Jeżeli *nazwa* została zdefiniowana, to zostanie pominięty fragment aż do najbliższej instrukcji else albo endif.

while – pętla warunkowa

Składnia while *wyrażenie*
. . .
endw

Opis Kompilacja linii pomiędzy while oraz endw jest powtarzana tak długo, dopóki *wyrażenie* jest prawdziwe. Wyrażenia o wartości różnej od zera są interpretowane jako prawdziwe; równe zero jako fałszywe.

Makrodefinicje

macro – deklaracja makrodefinicji

Składnia *nazwa* macro [*arg1*, *arg2*, ...]
. . .
endm

Opis Deklaruje sekwencję instrukcji jako makrodefinicję, która może być wstawiona w tekście źródłowym programu przez krótkie wywołanie. Makrodefinicja musi zostać zdefiniowana przed jej wywołaniem. Argumenty *arg1*, *arg2*,… są pobierane z miejsca wywołania.

Wywołanie *nazwa* [*arg1*, *arg2*, ...]

Komunikaty o błędach

error – wyprowadzenie komunikatu o błędzie

Składnia error "*łańcuch _znaków*"

Opis Wyprowadza komunikat o błędzie kompilacji w formacie identycznym jak komunikaty o błędach zdefiniowanych w MPASM. Używane zazwyczaj wewnątrz bloku if *wyrażenie*… endif w celu przerwania kompilacji w przypadkach, które nie zostały przewidziane w programie.

errorlevel – ustala sposób wyprowadzania komunikatów o błedach

Składnia errorlevel { 0 | 1 | 2 | +*nr_błędu* | -*nr_błędu* } [, ...]

Opis Określa, które rodzaje komunikatów mają być wyprowadzane.
0 – wszystkie komunikaty, ostrzeżenia i błędy,
1 – tylko ostrzeżenia i błędy,
2 – tylko błędy,
-*nr_błędu* – zabrania wyprowadzania komunikatu o podanym numerze,
+*nr_błędu* – umożliwia wyprowadzanie komunikatu o podanym numerze.

Przełączanie banków pamięci

Komendy asemblera dotyczące przełączania baków pamięci danych oraz programu wykazują duże podobieństwo do pseudoinstrukcji mikrokontrolera (Aneks A, Tabela A.5), jednakże dokumentacja do asemblera MPASM zalicza je do grupy komend asemblera.

Składnia	Opis	Równoważne operacje
BANKSEL adr	Ustawia bank pamięci danych dla adresowania bezpośredniego, który zawiera rejestr o adresie adr.	BCF / BSF 3, 5 BCF / BSF 3, 6
BANKISEL adr	Ustawia bank pamięci danych dla adresowania pośredniego (poprzez adres w rej. FSR), który zawiera rejestr o adresie adr.	BCF / BSF 3, 7
PAGESEL adr	Generuje kod wybierający bank w pamięci programu. Nowy bank zostanie uwzględniony w następującej dalej instrukcji goto lub call. Patrz także pseudoinstrukcje lgoto oraz lcall.	BCF / BSF 0x0A, 3 BCF / BSF 0x0A, 4

D. Mapy pamięci wybranych mikrokontrolerów z rodziny PIC16

	Adres		Adres
INDF	00h	INDF	80h
TMR0	01h	OPTION_REG	81h
PCL	02h	PCL	82h
STATUS	03h	STATUS	83h
FSR	04h	FSR	84h
PORTA	05h	TRISA	85h
PORTB	06h	TRISB	86h
	07h		87h
EEDATA	08h	EECON1	88h
EEADR	09h	EECON2	89h
PCLATH	0Ah	PCLATH	8Ah
INTCON	0Bh	INTCON	8Bh
GPR Rejestry uniwersalne, 68 bajtów SRAM	0Ch 4Fh	Dostęp do SRAM w banku 0	8Ch CFh
	50h		D0h
	7Fh		FFh
Bank 0		Bank 1	

Oznaczenia:

Niezaimplementowane komórki pamięci, odczytywane jako 0.

Tabela D.1. Mapa pamięci mikrokontrolera PIC16F84A.

	Adres		Adres		Adres		Adres
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
	07h		87h		107h		187h
	08h		88h		108h		188h
	09h		89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Rezerwacja (1)	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Rezerwacja (1)	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h		91h				
T2CON	12h	PR2	92h				
SSPBUF	13h	SSPADD	93h				
SSPCON	14h	SSPSTAT	94h				
CCPR1L	15h		95h				
CCPR1H	16h		96h				
CCP1CON	17h		97h				
	18h		98h				
	19h		99h				
	1Ah		9Ah				
	1Bh		9Bh				
	1Ch		9Ch				
	1Dh		9Dh				
ADRESH	1Eh	ADRESL	9Eh				
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
GPR Rejestry uniwersalne, 96 bajtów SRAM	20h … 7Fh	GPR Rejestry uniwersalne, 80 bajtów SRAM / Dostęp do 70h-7Fh	A0h … EFh / F0h … FFh	GPR Rejestry uniwersalne, 80 bajtów SRAM / Dostęp do 70h-7Fh	120h … 16Fh / 170h … 17Fh	Dostęp do 20h-7Fh	1A0h … 1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

Oznaczenia:

Niezaimplementowane komórki pamięci, odczytywane jako 0.

(1): rejestry zarezerwowane; nie zapisywać.

Tabela D.2. Mapa pamięci mikrokontrolera PIC16F819.

	Adres		Adres		Adres		Adres
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD	08h	TRISD	88h		108h		188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Rezerwacja (1)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Rezerwacja (1)	18Fh
T1CON	10h		90h	GPR Rejestry uniwersalne, 16 bajtów SRAM	110h	GPR Rejestry uniwersalne, 16 bajtów SAM	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch	CMCON	9Ch		11Ch		19Ch
CCP2CON	1Dh	CVRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
GPR Rejestry uniwersalne, 96 bajtów SRAM	20h	GPR Rejestry uniwersalne, 80 bajtów SRAM	A0h	GPR Rejestry uniwersalne, 80 bajtów SRAM	120h	GPR Rejestry uniwersalne, 80 bajtów SRAM	1A0h
			EFh		16Fh		1EFh
		Dostęp do 70h-7Fh	F0h	Dostęp do 70h-7Fh	170h	Dostęp do 70h-7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

Oznaczenia:

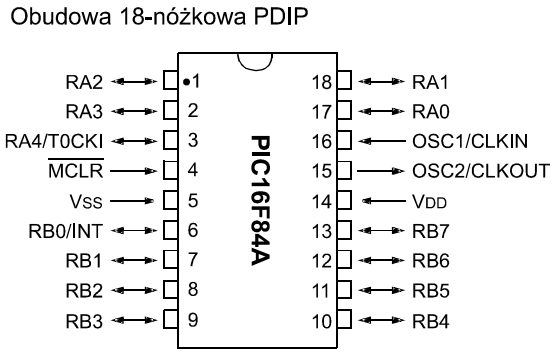
Niezaimplementowane komórki pamięci, odczytywane jako 0.

(1): rejestry zarezerwowane; nie zapisywać.

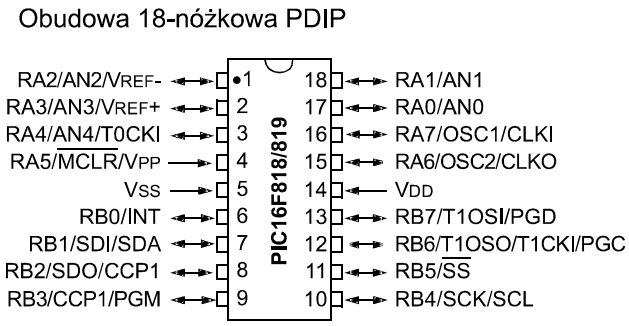
Tabela D.3. Mapa pamięci mikrokontrolera PIC16F877A.

E. Rozkład wyprowadzeń mikrokontrolerów

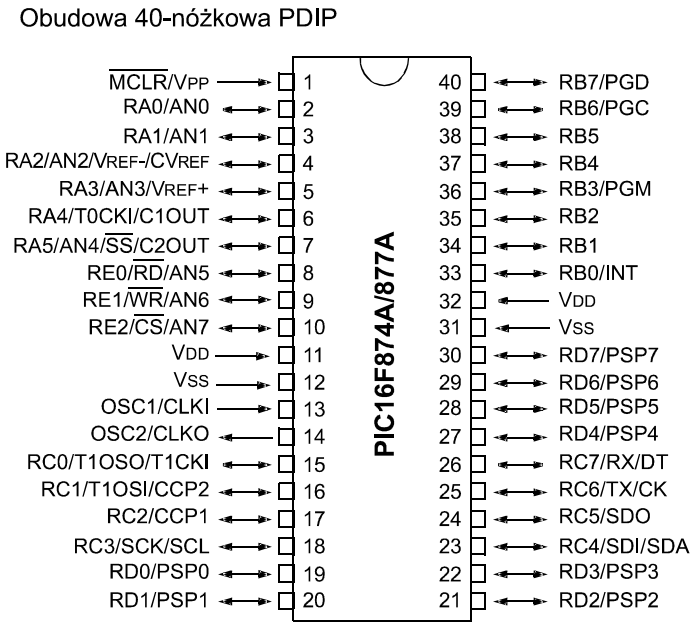
Mikrokontrolery PIC16 są produkowane w różnych obudowach, które nawet dla mikrokontrolerów tego samego typu różnią się liczbą i rozkładem wyprowadzeń. Mikrokontrolery dostępne w Laboratorium Elektroniki przeznaczone do pracy w zestawach uruchomieniowych ZL4PIC mają obudowy typu PDIP.



Rys. E.1. Rozkład wyprowadzeń mikrokontrolera PIC16F84A w obudowie PDIP.



Rys. E.2. Rozkład wyprowadzeń mikrokontrolerów PIC16F818 i 819 w obudowie PDIP.



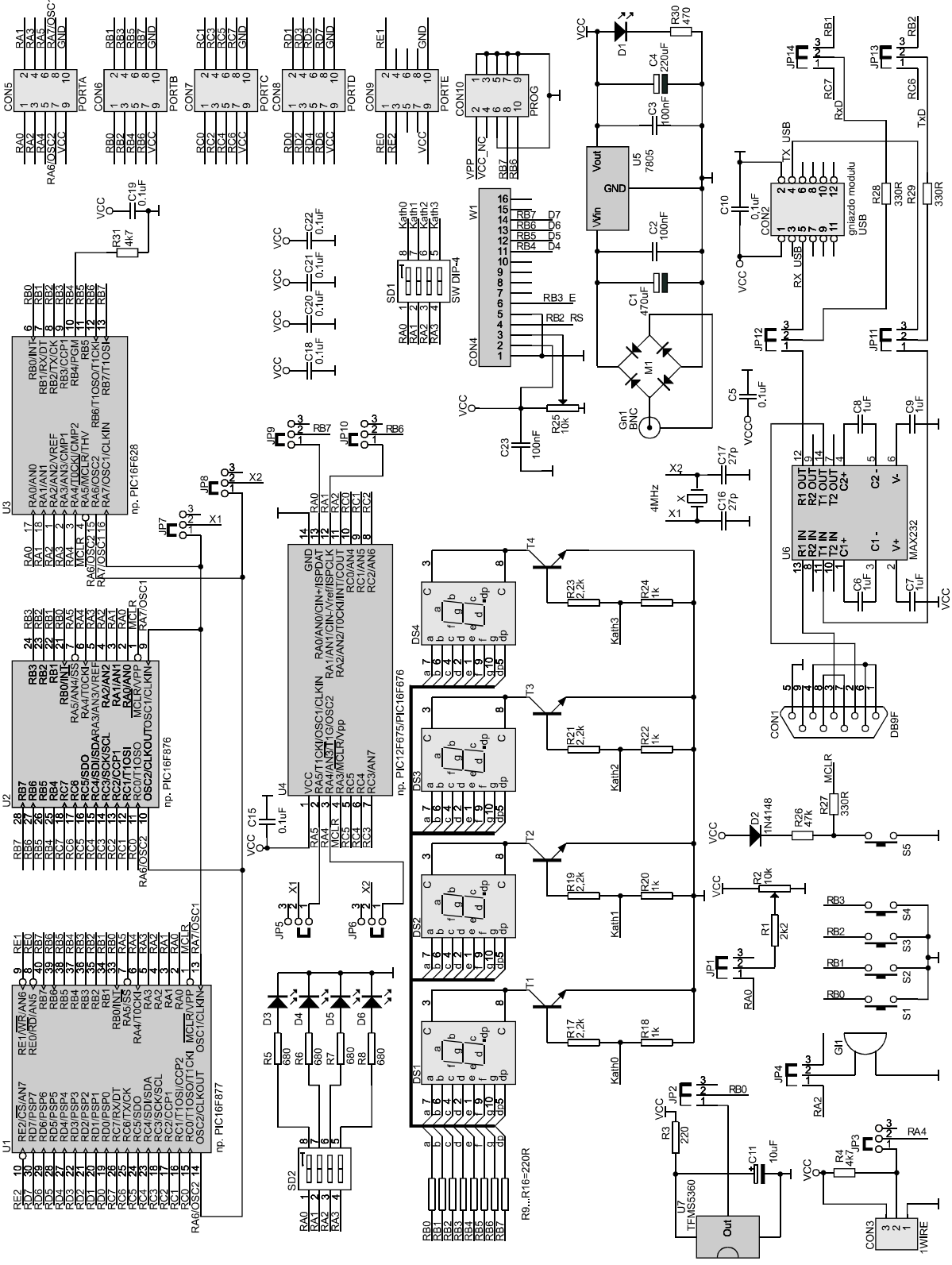
Rys. E.3. Rozkład wyprowadzeń mikrokontrolera PIC16F877A w obudowie PDIP.

Opis wybranych funkcji linii mikrokontrolerów:

AN0…AN7	–	wejścia analogowe,
CLKI(N)	–	wejście sygnału z zewnętrznego zegara,
CLKO(UT)	–	wyjście sygnału zegarowego z wewnętrznego oscylatora RC,
$\overline{\text{MCLR}}$	–	reset mikrokontrolera wyzwalany stanem niskim na tym wejściu,
PGM	–	linia zarezerwowana dla programatora gdy używane jest programowanie niskim napięciem,
RA0…RA7	–	linie portu A skonfigurowane jako cyfrowe wej./wyj. ogólnego przeznaczenia,
RB0…RB7	–	linie portu B skonfigurowane jako cyfrowe wej./wyj. ogólnego przeznaczenia,
RC0…RC7	–	linie portu C skonfigurowane jako cyfrowe wej./wyj. ogólnego przeznaczenia,
RD0…RD7	–	linie portu D skonfigurowane jako cyfrowe wej./wyj. ogólnego przeznaczenia,
RE0…RE2	–	linie portu E skonfigurowane jako cyfrowe wej./wyj. ogólnego przeznaczenia,
OSC1, OSC2	–	miejsca podłączenia rezonatora kwarcowego,
V_{DD}	–	dodatnie napięcie zasilania,
V_{PP}	–	miejsce przyłączenia napięcia +12...14 V z programatora w trybie wysokonapięciowym,
V_{SS}	–	masa.

F. Zestaw uruchomieniowy ZL4PIC

Zestaw uruchomieniowy ZL4PIC posiada odrębną instrukcję przygotowana przez producenta (BTC), która dostępna jest w dziale „Materiały pomocnicze” na stronie internetowej przedmiotu Technika Cyfrowa/zakładka „Laboratorium”. W niniejszej instrukcji ograniczmy się do podania schematu tego zestawu.



Rys. F.1. Schemat elektryczny zestawu ZL4PIC.