НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра автоматизації технологічних процесів і виробництв

Лабораторна робота № 5 з курсу ”Мікропроцесорні та програмні засоби автоматизації”

Виконання елементарних операцій на програмному симуляторі PIC Simulator IDE

Тернопіль 2018

Методичні вказівки до лабораторної роботи №5 “Виконання елементарних операцій на програмному симуляторі PIC Simulator IDE” з курсу «Мікропроцесорні та програмні засоби автоматизації» з курсу «Мікропроцесорні та програмні засоби автоматизації». Медвідь В.Р., Пісьціо В.П., - Тернопіль: ТНТУ, 2018 - 15 с.

Відповідальні за випуск доцент, к.т.н. Медвідь В.Р.,
асистент Пісьціо В.П.

Для студентів напряму: 151 "Автоматизація та комп'ютерно-інтегровані технології"

Лабораторна робота №5
Виконання елементарних операцій на програмному симуляторі PIC Simulator IDE

1. Послідовність роботи з програмним симулятором PIC Simulator IDE
Основне вікно програми PIC Simulator IDE має вигляд, показаний на (рис. 1).

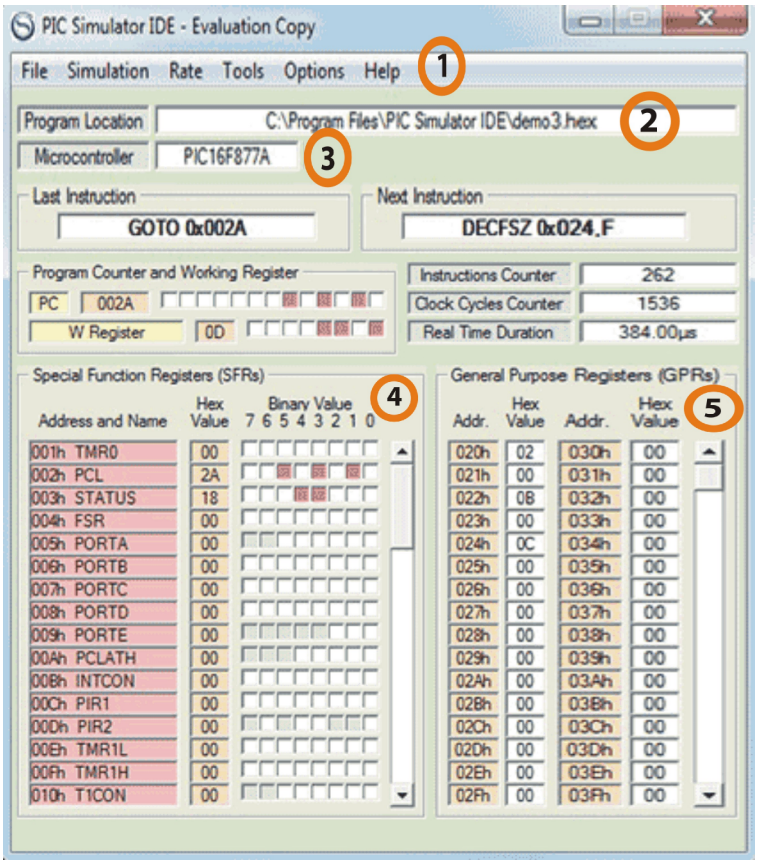

Рис. 1. Основне вікно програми PIC Simulator IDE

У верхній частині знаходяться меню, через які можна отримати доступ до основних і додаткових модулів програми (поз. 1)(рис. 1).

В рядку Program Location вказано шлях до обраної програми і її ім'я (поз. 2).

В рядку Microcontrollers, відображається тип обраного мікроконтролера (поз. 3).

У нижній частині вікна є дві панелі (поз.4 і поз.5), в яких відображаються стан програми, вміст регістрів спеціальних функцій (РСФ) і керуючих регістрів обраного МК.

Послідовність роботи з програмним симулятором наступний:

- запуск програми PIC Simulator IDE;
- вибір типу мікроконтролера, для якого написана програма;
- вибір частоти кварцового генератора (впливає тільки на відображувані програмою дані про час виконання програми або команди, але не на швидкість роботи програми, що налагоджуються в PIC Simulator IDE);
- завантаження програми у вигляді HEX-файлу або запуск вбудованого компілятора мови асемблера і написання в ньому потрібної програми;
- вибір потрібних модулів віртуальних пристроїв;
- вибір швидкості і режиму роботи програми симулятора;
- запуск процесу симуляції роботи програми на обраному МК.

Якщо потрібно скористатися для роботи з симулятором власною програмою або внести зміни у вже розроблену, необхідно створити або завантажити для цього файл асемблера, з якого після компіляції буде створений необхідний для роботи з симулятором hex-файл.

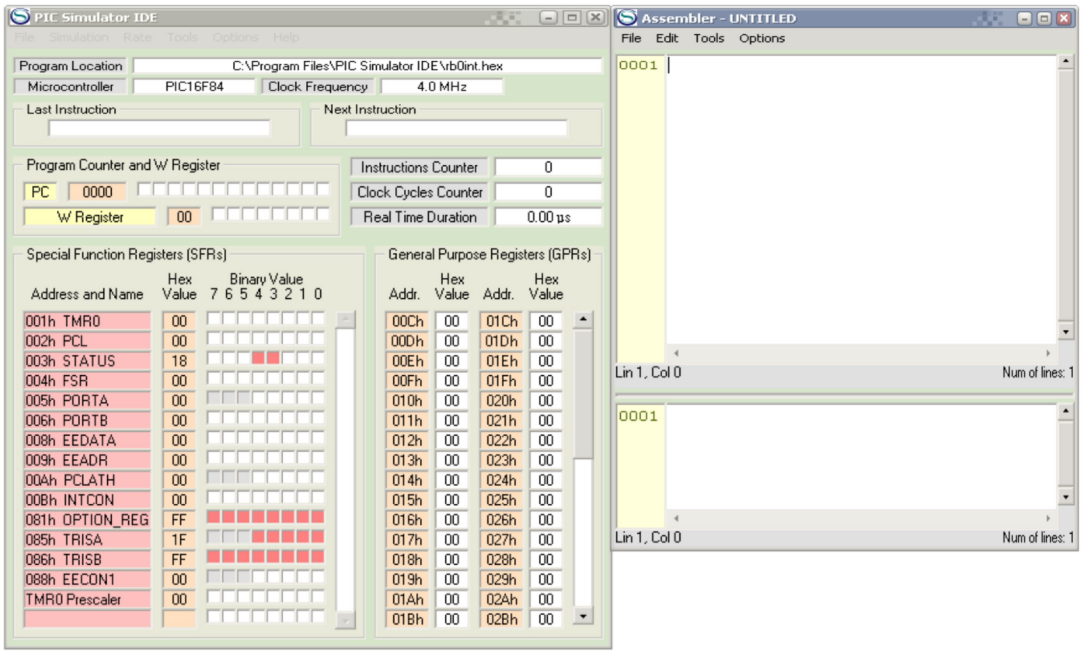

Рис. 2 Вікно симулятора з відкритим вікном Assembler

Для цього:

1. Натиснути Options | Assembler. Відкриється вікно компілятора Assembler – UNTITLED (рис. 2);

2. У вікні Assembler натиснути опцію File. Розкриється закладка (рис. 3), з якої для створення нового файлу потрібно натиснути New, а для завантаження вже створеного – OPEN.

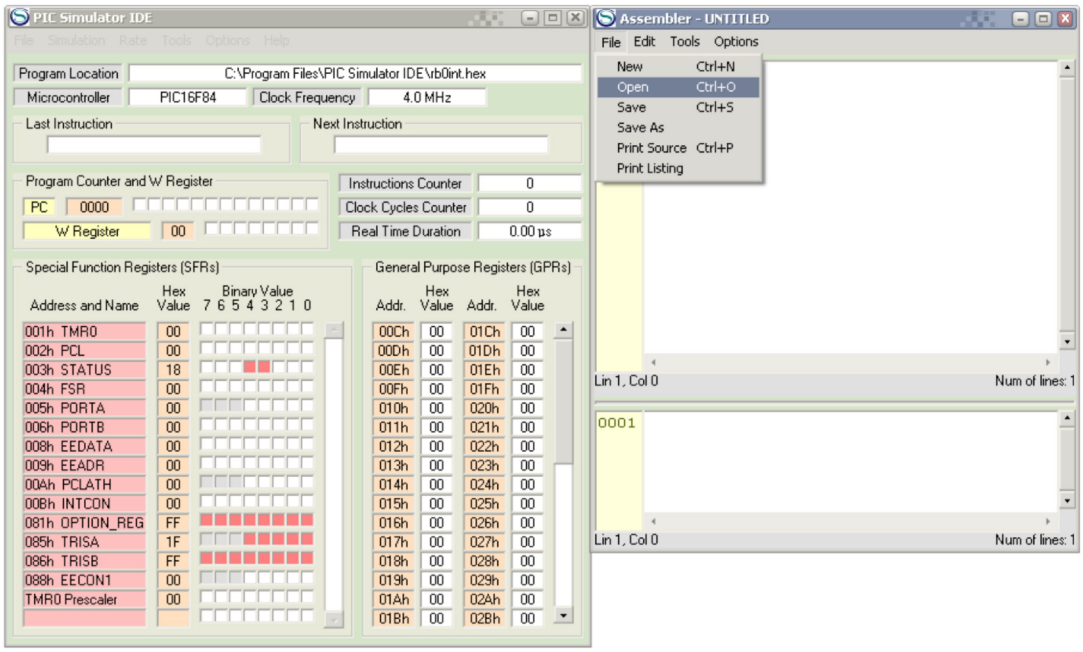

Рис. 3 Завантаження існуючого або створення нового файлу асемблера

3. Після вибору і завантаження файлу (наприклад, rb0int.asm), його текст з’явиться у вікні Assembler (рис. 4).

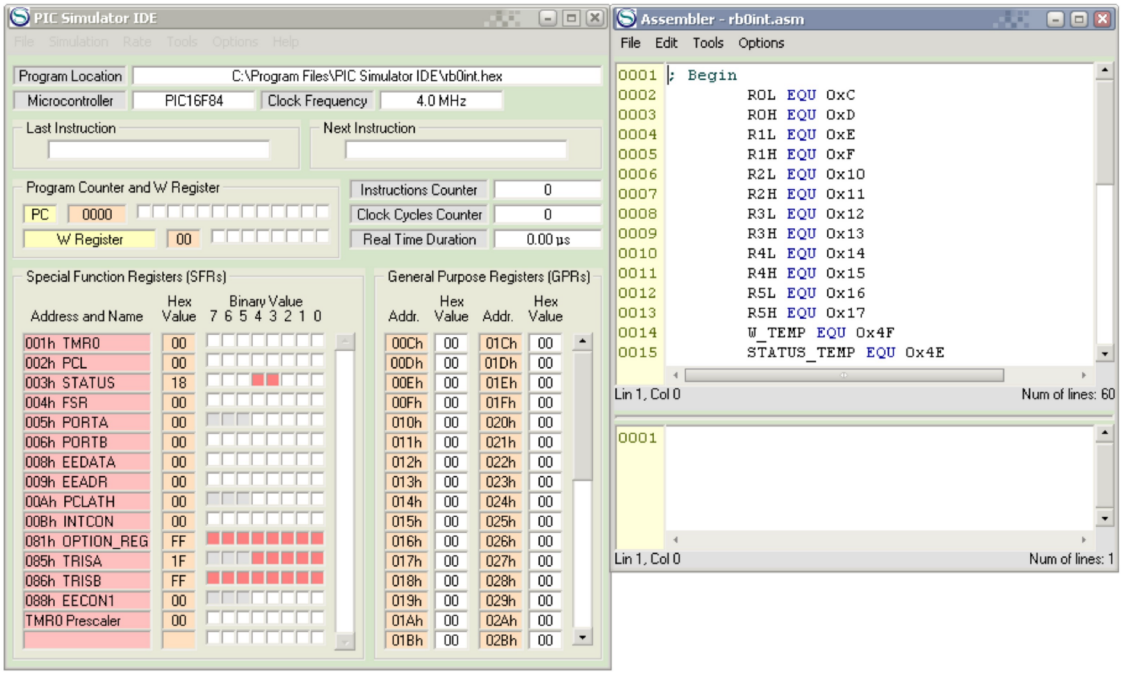

Рис. 4 Завантаження файлу rb0int.asm

4. Для компіляції створеного або завантаженого і потім зміненого файлу, натисніть Tools і у вікні, що розкриється – Assemble. В нижній половині вікна Assembler з'явиться лістинг відкомпільованого файлу і, одночасно, при відсутності помилок, буде створений одноіменний hex-файл.

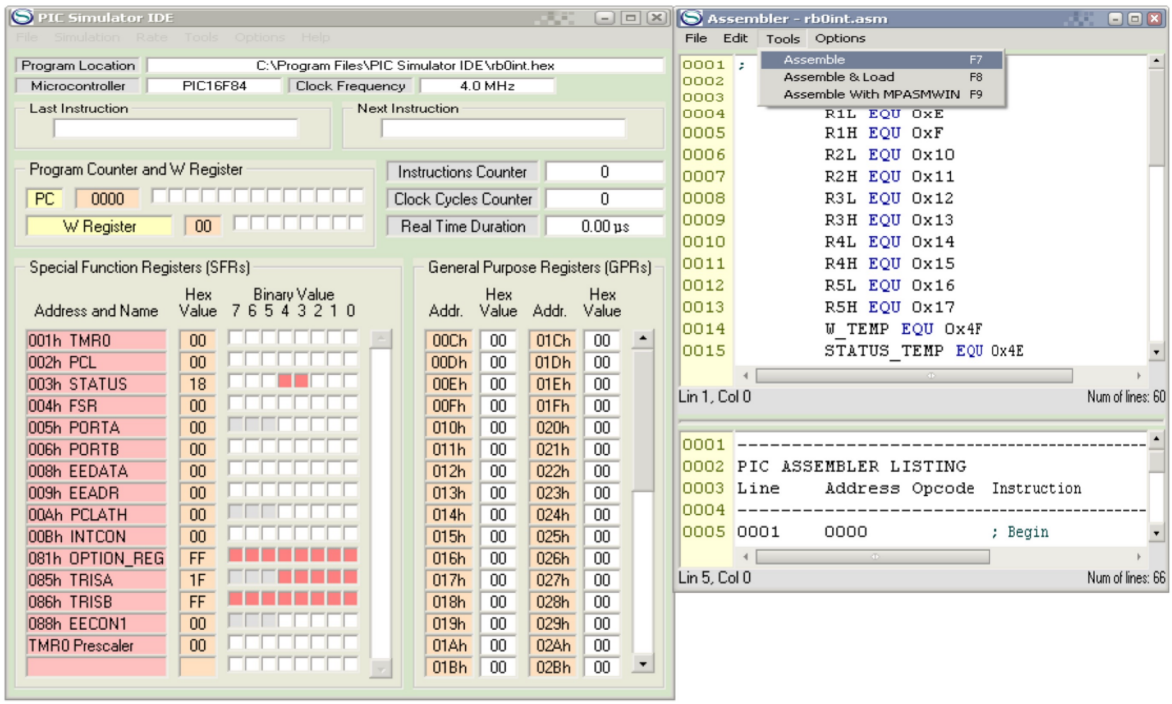

Рис. 5

2. Завдання на лабораторну роботу: математичні операції, підпрограма множення.

1. Вивчити програмну модель PIC Simulator IDE.
2. Вивчити команди арифметичних, логічних операцій, операцій з портами та лічильником

PIC – контролера.

3. Дослідити роботу програм за вказівкою викладача та вміст регістрів контролера (W, STATUS…), які використовуються при виконанні цієї програми.

4. Записати для вибраних команд асемблера коментар щодо їх призначення.

3. Послідовність роботи з симулятором при виконанні програм

Виконаємо програму відповідно до вказаного викладачем завдання в PIC Simulator ID, для чого необхідно:

1. Запустити PIC Simulator IDE;
2. Натиснути Options | Select Microcontroller;
3. Вибрати PIC16F84 і натиснути кнопку Select;
4. Натиснути Tools і у вікні, що розкриється, вибрати «Assembler». Відкриється вікно компілятора «Assembler – UNTITLED» (рис. 2);
5. Набрати текст заданої програми у вікні «Assembler»;
6. Натиснути Tools і у вікні, що розкриється – Assemble. В нижній половині вікна Assembler з'явиться лістинг відкомпільованого файлу (рис. 6);
7. Одночасно, при відсутності помилок, буде створений файл з розширенням «hex», для якого можна вибрати ім'я та шлях для запису. Записати його на «Робочий стіл» комп'ютера;
8. Вибрати File | Load Program і завантажити створений файл hex-файл;

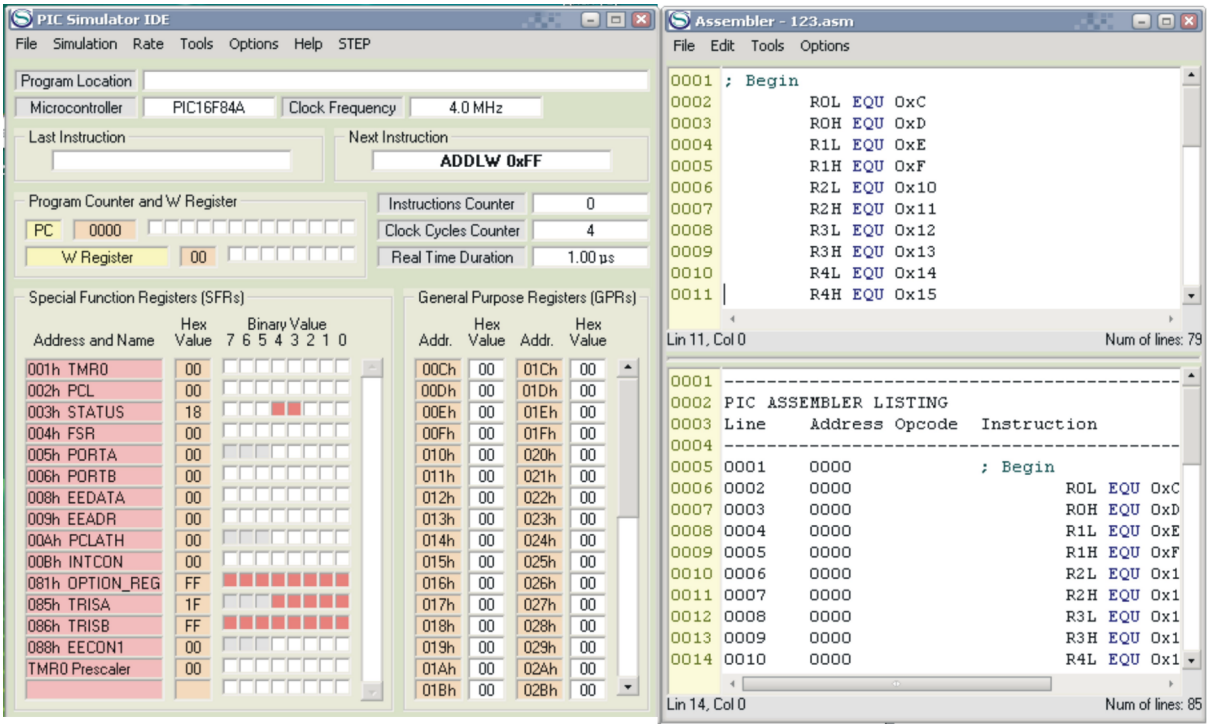

Рис. 6 Вигляд екрану з програмою

9. В основному вікні симулятора натиснути Rate | Step By Step, а далі вибрати опцію Simulation і натиснути Start. Симулятор готовий до виконання програми в кроковому режимі;

10. Для виконання наступної команди програми потрібно натиснути на закладку STEP, яка з'явиться справа від закладки HELP вгорі основного вікна симулятора після вибору крокового режиму його роботи;

11. Для виконання програми в автоматичному режимі потрібно вибрати Rate | Extremely Fast simulation rate;

12. Щоб зупинити виконання програми, потрібно натиснути Simulation | Stop.

Вміст регістрів контролера, які використовуються при виконанні команд програми, знайти в області регістрів Adress and Name, яка розташована в лівій нижній частині основного вікна симулятора (виділені рожевим кольором). Всі регістри восьмирозрядні.

В процесі виконання програми по зміні кольору комірок видно, вміст яких регістрів змінюється. Забарвлення комірки відповідного розряду регістру помаранчевим кольором означає наявність “1”, білим - “0”.

4. Особливості програмування на асемблері для PIC-контролера

Інформація до команд, що наведені в таблиці команд (табл. 3).

У першій колонці розташовується назва команди. Латинські літери в кінці команди означають:

f - назва регістра (в загальному вигляді), до якого ця команда звертається, і з вмістом якого буде проводитися дія. Тобто, наявність в команді літери f означає, що команда звертається до вмісту регістра, з яким програміст хоче провести визначену цією командою дію;

d - покажчик місця збереження результату при операціях з байтами:

- якщо d = 0, то результат виконання команди зберігається в регістрі W (в акумуляторі),
- якщо d = 1, то результат виконання команди зберігається в регістрі f;

b - номер біта в регістрі, з яким проводиться дія;

k - константа, тобто певне число, яке задається користувачем.

У другій колонці знаходиться опис дії, яка реалізується командою.

У третій колонці вказано кількість машинних циклів, протягом яких дана команда виконується. Більшість команд виконується за 1 машинний цикл.

Для команд, що виконуються за 2 машинних цикли, двійка знаходиться в дужках.

Комбінація числа 1 і двійки в дужках означає, що за одним сценарієм виконання команди вона виконується за один машинний цикл, а за іншим - за два.

У четвертій колонці вказані флажки, які реагують на виконання даної команди.

Значна частина команд складається з 5-ти літер, причому корінь слова команди, що характеризує її суть, займає, в більшості випадків, перші три літери, а останні дві показують, звідки і куди копіюється (переписується) число або з якими регістрами потрібно виконати визначену командою дію.

Наприклад, команду ADDWF можна умовно розділити на 2 частини: корінь ADD, який означає операцію додавання, і WF, де перша літера позначає регістр W (акумулятор), а друга - регістр F, тобто регістр, до якого звертається команда.

Отже, з самої назви команди стає зрозумілим, що додається вміст регістрів W і F.

У більшості випадків, в якості цього умовного регістра F використовуються регістри загального призначення, а також регістри спеціальних функцій.

Пряма і непряма адресація даних

Коли здійснюється **пряма адресація** даних, молодші 7 біт беруться як пряма адреса з коду операції, а два біти покажчика сторінок (RP1, RP0) - з регістра статусу.

Ознакою **непрямої адресації** є звертання до регістру **INDF**. Будь-яка команда, що використовує INDF (адресу 00h) як регістр, фактично звертається до покажчика, що зберігається в **FSR** (адреса 04h).

Читання непрямим чином самого регістра INDF дасть результат 00h. Непрямий запис у регістр INDF призведе до втрати даних, що записувались, хоч і може змінити біти стану.

Необхідна 9-бітна адреса формується об'єднанням вмісту 8-бітного FSR регістра і біту IRP з регістра статусу

5. Завдання на лабораторну роботу

Приклад 1. Додавання операндів

1.1. Додати вміст регістру R0L до вмісту акумулятора W і результат операції додавання записати в регістр R0H.

```
; Begin
    ORG 0x0000          ; визначаємо початкову адресу пам’яті для запису програми
    R0L EQU 0x20        ; регістр за адресою 20 позначаємо як R0L
```

```
        R0H EQU 0x21        ; регістр за адресою 21 позначаємо як R0H
        BCF PCLATH,3        ; скидаємо в нуль 3-й розряд програмного лічильника PCLATH
        BCF PCLATH,4        ; скидаємо в нуль 4-й розряд програмного лічильника PCLATH
        MOVLW 0x10          ; записуємо в регістр W шістнадцяткове число 10
        MOVWF R0L           ; пересилаємо в регістр R0L вміст W
        MOVLW 0x0A          ; записуємо в регістр W шістнадцяткове число 0A
        ADDWF R0L,0         ; додаємо вміст W та R0L, результат записуємо в W
        MOVWF R0H           ; пересилаємо вміст акумулятора в R0H
END
```

1.2. Додати вміст регістру R0L до вмісту акумулятора W і результат операції додавання записати в W.

```
; Begin
        ORG 0x0000
        R0L EQU 0x20
        BCF PCLATH,3
        BCF PCLATH,4
        MOVLW 0x16
        MOVWF R0L
        MOVLW 0x1A
        ADDWF R0L,0             ; додаємо вміст W та R0L, результат записуємо в W
END
```

1.3. Додати вміст регістру R0L до вмісту акумулятора W і результат операції додавання записати в R0L.

У цьому випадку команду **ADDWF R0L,0** потрібно записати у вигляді **ADDWF R0L,1** (або **ADDWF R0L,1**). Решту команд програми залишаємо без змін.

Завдання. Виконуються в лабораторії

1. Виконати програми Прикладу 1 (1.1, 1.2 та 1.3) на програмному симуляторі в кроковому режимі його роботи.
2. Записати вміст регістрів **R0L, R0H, W** в кінці виконання кожної з програм.

Приклад 2. Віднімання операндів. Визначення знаку результату по значенню ознаки C

Для команди **ADDWF** цей біт встановлюється в «1», якщо в результаті операції із старшого (сьомого) розряду відбувається перенесення.

Віднімання **SUBWF** здійснюється шляхом додавання додаткового коду другого операнду.

При виконанні команд зсуву біт C завжди завантажується з молодшого або старшого біту оперенда, що зсувається.

2.1. Підпрограма віднімання операндів. Результат додатній

```
; Begin
         ORG 0x0000
         R0L EQU 0x20
          MOVLW 0x10
         MOVWF R0L
         MOVLW 0x05
         SUBWF R0L,0            ; результат від'ємний, розміщується в W
END
```

2.1. Підпрограма віднімання операндів. Результат від'ємний

```
; Begin
        ORG  0x0000
        R0L  EQU 0x20
```

```
        MOVLW  0x05
        MOVWF  0x10
        SUBWF  R0L,1          ; результат додатній, розміщується в R0L
END
```

Завдання. Виконуються в лабораторії

1. Виконати програми Прикладу 2 (2.1 та 2.2) на програмному симуляторі в кроковому режимі його роботи.
2. Записати вміст регістрів **R0L, W** та **C** в кінці виконання кожної з програм.
3. З вікна з програмою (рис. 6) для команд за таблицею команд асемблера для PIC-контролера (табл. 3) записати коментар щодо призначення цих команд відповідно до наведеного нижче прикладу:

Код команди	**Команда**	**Виконувана операція (коментар)**
118A	BCF PCLATH, 3	; скинути в “0” 3-ій біт регістра PCLATH

і т.д.

Приклад 3. Логічні операції

3.1. Виконати логічне АБО вмісту регістра R0L та вмісту акумулятора W і результат операції записати в W.

```
; Begin
      ORG 0x0000
      R0L EQU 0x20
      BCF PCLATH,3
      BCF PCLATH,4
      MOVLW 0x09
      MOVWF R0L
      MOVLW 0x11
      IORWF R0L,0        ; логічне додавання вмісту W та R0L, результат записуємо в W
END
```

3.2. Виконати логічне І вмісту регістра R0L та вмісту акумулятора W і результат операції записати в R0L.

```
; Begin
      ORG 0x0000
      R0L EQU 0x20
      BCF PCLATH,3
      BCF PCLATH,4
      MOVLW 0x09
      MOVWF R0L
      MOVLW 0x11
      ANDWF R0L,1      ; логічне додавання вмісту W та R0L, результат записуємо в W
END
```

Завдання. Виконуються в лабораторії

1. Виконати програми Прикладу 3 (3.1 та 3.2) на програмному симуляторі в кроковому режимі його роботи.
2. Записати вміст регістрів **R0L, W** та **C** в кінці виконання кожної з програм.
3. Записати коментар до кожної з команд підпрограми.

Приклад 4. Запис в пам’ять EEPROM

Необхідно спочатку записати потрібну адресу в EEADR регістр і дані в EEDATA регістр. Далі виконати специальну послідовність команд, що виконує безпосередній запис:

```
; Begin
        ORG  0x0000
        BCF   STATUS,RP0
        MOVLW 0x00
        MOVWF EEADR
        MOVLW 0x10
        MOVWF EEDATA
        BSF   STATUS,RP0
        MOVLW 55h
        MOVWF EECON2
        MOVLW AAh
        MOVWF EECON2
        BSF EECON1,WR          ; встановити біт WR, почати запис
END
```

Завдання. Виконуються в лабораторії

1. Виконати програми Прикладу 4 на програмному симуляторі в кроковому режимі його роботи.
2. Записати вміст регістрів EECON2 та EECON1 в кінці виконання підпрограми.
3. Записати коментар до кожної з команд підпрограми.

Приклад 5. Підпрограма для запису в 15 регістрів (діапазон адрес 20h - 2Fh) числа 02 з використанням непрямої адресації (адреса знаходиться в регістрі FSR):

```
; Begin
        ORG  0x0000
        BCF STATUS, IRP        ; встановити банки 0 і 1
        MOVLW 20h              ; вказуємо перший регістр, що очищається
        MOVWF FSR
M1:
                               ; CLRF INDF  - очистити регістр INDF
        BSF INDF,1             ; записати в регістра INDF число 2
        INCF FSR, F            ; збільшити адресу
        BTFSS FSR, 4           ; перевірка
        GOTO M1                ; продовжити очищення
END
```

Для очищення 15-ти регістрів (діапазон адрес h20 - h2F, запис «00») з використанням непрямої адресації **команду BSF INDF,1 потрібно замінити командою CLRF INDF.**

Завдання. Виконуються в лабораторії

1. Виконати програми Прикладу 4 на програмному симуляторі в кроковому режимі його роботи.
2. Записати вміст регістрів FSR, STATUS, регістрів за адресами 20h - 24h та W в кінці виконання програми.

Приклад 6. Підпрограма реалізації затримки на 10 машинних циклів (10 мкс):

```
; Begin
        ORG  0x0000
         REG1 EQU 20h          ; визначаємо адресу регістра під число
        MOVLW  3               ; записуємо число 3 в акумулятор W
        MOVWF REG1             ; копіюємо з акумулятора в Reg1
                               ; реалізація самої затримки:
```

```
WRITE:
        DECFSZ REG1,1           ; зменшити Reg1 на одиницю і результат зберегти Reg1
        GOTO WRITE              ; перехід за міткою write
        MOVLW  0A
END
```

Приклад 7. Програмування ліній портів на ввід та вивід

```
; Begin
          ORG  0x0000
          BSF STATUS,RP0
          MOVLW 0x80
          MOVWF OPTION_REG
          MOVLW 0x0F
          MOVWF TRISB
          BCF STATUS,RP0
          MOVLW 0x1A
          MOVWF PORTB
END
```

Завдання. Виконуються в лабораторії

1. Виконати програму на програмному симуляторі в кроковому режимі його роботи.
2. Записати вміст регістрів STATUS, TRISB та PORTB в кінці виконання підпрограми.
3. Записати коментар до кожної з команд підпрограми.

Приклад 8. Реалізація часової затримки з використанням таймера

Процесор виконує передбачені алгоритмом команди, а таймер одночасно з цим відраховує задану затримку. Після її завершення (переповнення таймера з 'FF' до '00') автоматично встановлюється в "1" флажок T0IF, який може бути перевірений програмно.

А встановивши попередньо "1" в біті дозволу переривання T0IE, можна після відрахованої таймером затримки автоматично перейти до підпрограми обробки переривання.

Регістри, пов'язані з його роботою, подані в табл. 1.

Передподільник може бути під'єднаний до TMR0, якщо встановити PSA = 0, або до сторожового таймера WDT (PSA = 1). Коефіцієнт ділення передподільника задається бітами PS2…PS0 (табл.2).

Табл. 1. Регістри, пов'язані з роботою таймера TMR0

Адреса	Назва	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
0x81	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
0x0B	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Коли біт T0CS = 0, TMR0 працює від внутрішнього тактового генератора, збільшуючи свій вміст на одиницю після кожних Кп машинних циклів.

Якщо біт T0CS = 1, джерелом імпульсів стає пристрій, підключений ззовні до входу мікроконтролера RA4. Активний фронт цього сигналу визначає біт T0SE.

Табл. 2. Формат регістру конфігурації (OPTION)

Назва	Позиція	Призначення
$\overline{\text{RBPU}}$	7	Біт установки підтягуючих «pull-up» резисторів на виводах PORTB: 0 = резистори підключені; 1 = резистори відключені.

Табл. 2. Формат регістру конфігурації (OPTION)

Назва	Позиція	Призначення		
INTEDG	6	Біт вибору переходу сигналу переривання: 0 = переривання по спаду сигналу на виводі RBO/INT; 1 = переривання по фронту сигналу на виводі RBO/INT.		
TOCS	5	Біт вибору джерела сигналу таймера TMRO: 0 = внутрішній тактовий сигнал (CLKOUT); 1 = перехід на виводі RA4/TOCKI.		
TOSE	4	Біт вибору переходу джерела сигналу для TMR0: 0 = збільшення по фронту сигналу на виводу RA4/TOCKI; 1 = збільшення по спаду сигналу на виводі RA4/TOCKI.		
PSA	3	Біт призначення подільника: 0 = передподільник підключений до TMR0; 1 = передподільник підключений до сторожового таймера WDT.		
PS2- PS0	2-0	Біти вибору коефіцієнта розподілу передподільника.		
		Значення бітів	Швидкість для TMR0	Швидкість для Watchdog таймера
		000	2	1:1
		001	4	1:2
		010	8	1:4
		011	16	1:8
		100	32	1:16
		101	64	1:32
		110	128	1:64
		111	256	1:128

Використання передподільника дозволяє значно розширити діапазон можливих затримок таймера TMR0. Максимальний коефіцієнт поділу тактової частоти складає 256.

Алгоритм програми передбачає підключення передподільника до таймера TMR0 з коефіцієнтом ділення частоти Кп = 256 (або 2^8).

Оскільки при частоті генератора 4 МГц машинний цикл триває 1 мкс (чотири такти), **переповнення таймера настає через**

$$T_{TMR0} = Кп \times (256 - N) \text{ мкс} \qquad (1)$$

де N – число, яке потрібно записати в таймер.

Нехай, потрібно реалізувати затримку в 20 мкс (або 20 машинних циклів), за допомогою таймера.

Тоді, за виразом (1) при Кп=2, маємо

$20 = 2 \times (256 - N)$ мкс

Звідки знайдемо N=246 (F6h).

Реалізуємо часову затримку програмою:

```
; Begin
        ORG  0x0000
        BSF STATUS,RP0
        MOVLW 0x80
        MOVWF OPTION_REG
        BCF STATUS,RP0
        BCF INTCON,GIE              ; заборона переривань
        BCF INTCON,T0IF             ; скидання прапорця переповнення таймера
        MOVLW 246                   ; формування константи для
        MOVWF TMR0                  ; ініціалізації таймера
M1:
        BTFSS  INTCON,T0IF          ; очікування переповнення таймера
        GOTO M1                     ; перехід, якщо таймер не переповнений
```

```
                                ; продовження програми
    MOVWF 20h                   ; після скидання лічильника записуємо вміст W в
                                ; регістр за адресою 20h.
END
```

Завдання. Виконуються в лабораторії

1. Виконати програму на програмному симуляторі в режимі «NORMAL» та кроковому режимі його роботи.

2. Записати вміст регістрів OPTION_REG, TMR0 та регістра з адресою 20h в кінці виконання підпрограми.

3. Вирахувати максимально можливу затримку з використанням таймера для тактової частоти 4 МГц.

6. Контрольні запитання

1. Використання PIC-контролерів.
2. Призначення регістра W мікроконтролера.
3. Арифметичні та логічні команди мікроконтролера.
4. Програмування портів PIC-контролера.
5. Програмування таймера-лічильника.
6. Формат регістрів STATUS та OPTION_REG.
7. Призначення та позначення основних елементів програмної моделі мікроконтролера.

5. Література

1. Данилин А. Программа-симулятор PIC Simulator IDE / Данилин А. // Современная электроника. 2006.- №4. -С. 68-76.

2. Тавернье К. PIC-микроконтроллеры. Практика применения. М.: ДМК, 2002.

3. Предко М. Создайте робота своими руками на PIC- контроллере./ Майкл Предко; Пер. с английского Земского Ю.В. – М.: ДМК Пресс, 2006. – 408 с.: ил.

4. Кениг А. и М. Полное руководство по PIC-микроконтроллерам.: Пер. с нем.-К.: МК-Пресс”, 2007.-256 с., ил.

Таблиця 3. Команди контролера PIC16X8X

Позначення	Функція	Цикли	Код команди	Біти стану	Прим.
ADDLW	Додавання константи і W	1	11111x	C, DC, Z	
ADDWF	Додавання W с f	1	00 0111 dfff ffff	C, DC, Z	1, 2
ANDLW	Логічне І константи і W	1	11 1001 kkkk kkkk	Z	
ANDWF	Логічне І W і f	1	00 0101 dfff ffff	Z	1, 2
BCF	Скидання біту в регістрі f	1	01 00bb bfff ffff		1, 2
BSF	Встановлення біту в регістрі f	1	01 01bb bfff ffff		1, 2
BTFSC	Пропустити команду, якщо біт у f дорівнює нулю	1 (2)	01 10bb bfff ffff		3
BTFSS	Пропустити команду, якщо біт у f дорівнює одиниці	1 (2)	01 11bb bfff ffff		3
CALL	Виклик підпрограми	2	10 0kkk kkkk kkkk		
CLRF	Скидання регістру f	1	00 0001 1fff ffff	Z	2
CLRW	Скидання регістра W	1	00 0001 0xxx xxxx	Z	
CLRWDT	Скидання сторожового таймера WDT	1	00 0000 0110 0100		
COMF	Інверсія регістру f	1	00 1001 dfff ffff	Z	1, 2
DECF	Декремент регістру f	1	00 0011 dfff ffff	Z	1, 2
DECFSZ	Декремент f, пропустити команду, якщо 0	1 (2)	00 1011 dfff ffff		1, 2, 3
GOTO	Перехід за адресою	2	10 1kkk kkkk kkkk		

INCF	Інкремент регістру f	1	00 1010 dfff ffff	Z	1, 2
INCFSZ	Інкремент f, пропустити команду, якщо 0	1 (2)	00 1111 dfff ffff		1, 2, 3
IORLW	Логічне АБО константи і W	1	11 1000 kkkk kkkk	Z	
IORWF	Логічне АБО W і f	1	00 0100 dfff ffff	Z	1, 2
MOVF	Пересилання регістру f	1	00 1000 dfff ffff	Z	1, 2
MOVLW	Пересилання константи в W	1	11 00xx kkkk kkkk		
MOVWF	Пересилання W у f	1	00 0000 1fff ffff		
NOP	Пуста команда	1	00 0000		
OPTION	Завантаження регістру OPTION	1	00 0000 0110 0010		
RETFIE	Повернення з переривання	2	00 0000 0000 1001		
RETLW	Повернення з підпрограми з заванта-женням константи в W	2	11 01xx kkkk kkkk		
RETURN	Повернення з підпрограми	2	00 0000 0000 1000		
RLF	Зсув f вліво через перенесення	1	00 1101 dfff ffff	C	1, 2
RRF	Зсув f вправо через перенесення	1	00 1100 dfff ffff	C	1, 2
SLEEP	Перехід у режим SLEEP	1	00 0000 0110 0011		
SUBLW	Вирахування W з константи	1	11 110x kkkk kkkk	C, DC, Z	
SUBWF	Вирахування W з f	1	00 0010 dfff ffff	C, DC, Z	1, 2
SWAPF	Обмін місцями тетрад в f	1	00 1110 dfff ffff		1, 2
TRIS	Завантаження регістру TRIS	1	00 0000 0110 0fff		
XORLW	Виключаюче АБО константи і W	1	11 1010 kkkk kkkk	Z	
XORWF	Виключаюче АБО W і f	1	00 0110 dfff ffff		1, 2

*Якщо в результаті виконання команди змінюється лічильник команд, або виконується перехід по перевірці умови, то команда виконується за два цикли. Другий цикл виконується як NOP.